Architectures matérielles numériques intégrées et réseaux de neurones à codage parcimonieux

Hugues Gerald Nono Wouafo

▶ **To cite this version:**

Hugues Gerald Nono Wouafo. Architectures matérielles numériques intégrées et réseaux de neurones à codage parcimonieux. Intelligence artificielle [cs.AI]. Université de Bretagne Sud, 2016. Français. NNT : 2016LORIS394 . tel-01332180

HAL Id: tel-01332180

https://theses.hal.science/tel-01332180

Submitted on 15 Jun 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE / UNIVERSITE DE BRETAGNE-SUD
sous le sceau de l'Université Bretagne Loire

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE DE BRETAGNE-SUD

Mention : Sciences et Technologies de l'Information et de la Communication
Ecole doctorale SICMA

Présentée par
Hugues Gérald NONO WOUAFO
Préparée au Lab-STICC, Lorient
Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance

Architectures matérielles numériques intégrées et réseaux de neurones à codage parcimonieux

Thèse soutenue le 15/01/2016
devant le jury composé de :

Bernard GIRAU
Pr., LORIA, Université Henri Poincaré Nancy 1, *Examinateur*

Benoît MIRAMOND
Pr., LEAT, Université de Nice, *Rapporteur*

Fan YANG
Pr., Le2i, Université de Bourgogne, *Rapporteur*

Philippe COUSSY
Pr., Lab-STICC, Université de Bretagne Sud, *Directeur de thèse*

Cyrille CHAVET
Dr., Lab-STICC, Université de Bretagne Sud, *Encadrant de thèse*

Avant-propos

Cette thèse rentre dans le cadre d'un contrat doctoral au laboratoire Lab-STICC de l'Université Bretagne Sud.

Remerciements

Je tiens à remercier,

Ma mère, mon père pour tout ce qu'ils ont fait, font et continueront de faire pour moi. PAPA, MAMAN, je continuerais à tout faire pour sauver vos ordinateurs, mais aussi à tout faire pour vous rendre fier. Merci encore pour tout.

Mes encadrants de thèse, pour toutes ces discussions passionnantes sur l'œil humain, pour leur confiance, leur aide, leurs conseils, et pour leur patience extraordinaire malgré mes défauts,

DANILO Robin, pour sa disponibilité, pour toute l'aide qu'il m'a apportée durant ma thèse, pour tous ses encouragements et pour tous ses conseils,

Les membres du jury, pour avoir accepté d'évaluer ces trois années de travaux,

Mes ainés et leur famille, son excellence M. DJOMO KAMGA Honoré, M. TAMO André, Professeur KAGO Innocent et Mme KAGO Marie Louise, au Docteur et à Mme KAMGA Thérèse, Mme KAKAGNE Hélène, qui m'ont beaucoup supportés et inspirés par leur gentillesse, par leur sagesse et par leur courage,

Mes cousins, cousines, grands frères, grandes sœurs et leur famille NDAYO Gaëlle, KAMGA Josselin, Willy, Cyrille, KAGO Daniel, Lionel, Fabrice, Eric qui m'ont beaucoup soutenus, conseillés et inspirés,

Toutes la famille MAURICET pour m'avoir accueilli dans leur maison, pour m'avoir donné du courage et tellement d'autres choses,

MADIBA Olivier –Sama (KIROO Games), pour une infinité de choses,

Les secrétaires, Florence et Virginie, pour cette belle ambiance, cette bonne humeur que vous donnez au laboratoire et pour toutes ces discussions très, très intéressantes,

Tous mes amis et collègues, Erwann, Charly, Claudia, Hervé, Jean-Pierre, David, Armelle, Ophélie, Athéna-Sama, Marion, Kamande, Hellerisse, Yoann (…) et toutes les autres personnes chères à mes yeux, pour votre présence, votre support et pour tous ces moments inoubliables que j'ai passé avec vous,

Tous ceux que j'ai côtoyés et qui m'ont encouragés, tous les shinobis et kunoichis de Kiro'o Games Studio qui portent notre beau rêve,

Mes erreurs, qui me rappellent que je suis imparfait mais aussi que je peux toujours faire mieux,

A Hermès Trismégiste et à la vie qui nous apprennent tout,

Au Principe primordial qui soutient tout ce qui existe et à son magnifique sens l'humour….

Résumé

De nos jours, les réseaux de neurones artificiels sont largement utilisés dans de nombreuses applications telles que le traitement d'image ou du signal. Récemment, un nouveau modèle de réseau de neurones a été proposé pour concevoir des mémoires associatives, le GBNN (Gripon-Berrou Neural Network). Ce modèle offre une capacité de stockage supérieure à celle des réseaux de Hopfield lorsque les informations à mémoriser ont une distribution uniforme. Des méthodes améliorant leur performance pour des distributions non-uniformes ainsi que des architectures matérielles mettant en œuvre les réseaux GBNN ont été proposées. Cependant, ces solutions restent très coûteuses en ressources matérielles, et les architectures proposées sont restreintes à des réseaux de tailles fixes et sont incapables de passer à l'échelle.

Les objectifs de cette thèse sont les suivants : (1) concevoir des modèles inspirés du modèle GBNN et plus performants que l'état de l'art, (2) proposer des architectures moins coûteuses que les solutions existantes et (3) concevoir une architecture générique configurable mettant en œuvre les modèles proposés et capable de manipuler des réseaux de tailles variables.

Les résultats des travaux de thèse sont exposés en plusieurs parties. Le concept de *réseaux à clones de neurone* et ses différentes instanciations sont présentés dans un premier temps. Ces réseaux offrent de meilleures performances que l'état de l'art pour un coût mémoire identique lorsqu'une distribution non-uniforme des informations à mémoriser est considérée. Des optimisations de l'architecture matérielle sont ensuite introduites afin de fortement réduire le coût en termes de ressources. Enfin, une architecture générique capable de passer à l'échelle et capable de manipuler des réseaux de tailles variables est proposée.

Abstract

Nowadays, artificial neural networks are widely used in many applications such as image and signal processing. Recently, a new model of neural network was proposed to design associative memories, the GBNN (Gripon-Berrou Neural Network). This model offers a storage capacity exceeding the one issued from Hopfield networks when the information to be stored has a uniform distribution. Methods improving performance for non-uniform distributions and hardware architectures implementing the GBNN networks were proposed. However, on one hand, these solutions are very expensive in terms of hardware resources and on the other hand, the proposed architectures can only implement networks with fixed sizes and they are not scalable.

The objectives of this thesis are: (1) to design GBNN inspired models outperforming the state of the art, (2) to propose architectures cheaper than existing solutions and (3) to design a generic architecture implementing the proposed models and able to handle various sizes of networks.

The results of these works are exposed in several parts. Initially, the concept of *clone based neural networks* and its variants are presented. These networks offer better performance than the state of the art for the same memory cost when a non-uniform distribution of the information to be stored is considered. The hardware architecture optimizations are then introduced to significantly reduce the cost in terms of resources. Finally, a generic scalable architecture able to handle various sizes of networks is proposed.

Table des matières

Liste des figures

CHAPITRE I. Contexte

Le cerveau humain est une machine performante capable de réaliser des opérations complexes telles que le raisonnement pour la résolution de problèmes, le contrôle moteur d'un membre ou encore la reconnaissance d'objets observés dans le champ visuel. Cet organe est le centre du système nerveux de l'être humain. L'énorme capacité du cerveau en termes de fonctionnalités ainsi que son efficacité énergétique sont les principales propriétés qui intéressent la communauté de la technologie de l'information.

Un des plus grand challenges du 21e siècle [1][2] consiste à comprendre le principe de fonctionnement du cerveau en reproduisant le comportement des réseaux de neurones biologiques du système nerveux et en les analysant à l'aide de modèles de réseaux de neurones artificiels. Dans ce champ de recherche, les neuroscientifiques ont exploré différents modèles mathématiques du fonctionnement du cerveau. Ces études ont permis l'introduction de réseaux de neurones pour réaliser de nombreuses applications [3] (ex : diagnostic médical [4]–[6], reconnaissance de l'écriture manuscrite [7]–[9], reconnaissance d'objets [10], prédiction financière [11]–[14] …). Elles ont aussi conduit à différentes mises en œuvre [15] telles que des circuits numériques [16], [17], des systèmes neurobiologiques [18]–[20] ou des algorithmes inspirés par le cerveau [21]–[23]. La Figure I.1 montre un aperçu des différentes mises en œuvre inspirées par l'étude de cet organe.

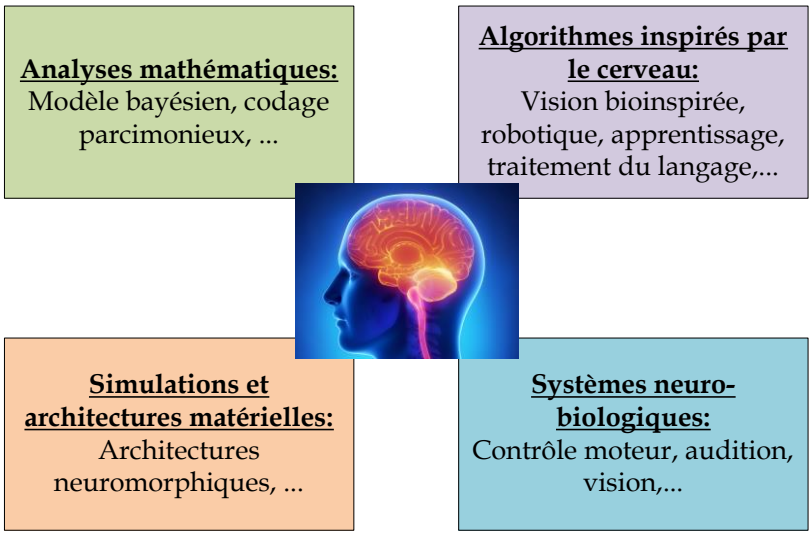


Figure I.1 Principales mises en œuvre basées sur les études du cerveau. Schéma inspiré de [14]. Image du cerveau issue de [24]

A. Le cerveau : un système complexe

A la différence d'un ordinateur conventionnel qui utilise des instructions programmées, le cerveau fonctionne grâce à un procédé complexe d'apprentissage, de stockage et de restitution de l'information [25]. En effet, le cerveau est un réseau extrêmement complexe de nœuds (neurones) et de connexions entre ces nœuds (synapses). Les connexions sont en majorité construites durant l'enfance et évolue pendant toute la vie. Les informations sont apprises en perdant ou en renforçant des connexions, principe qui est résumé sous le nom de *plasticité synaptique* [26]. Le cerveau crée des représentations invariantes des entrées diverses à travers une mémoire associative ayant une organisation hiérarchique complexe. A l'intérieur du cerveau, le traitement de l'information est un processus qui repose énormément sur la correspondance de motifs (*pattern matching*) et l'association sensorielle [27].

L'exploration du calcul bio-inspiré requiert des modèles formels d'organisation des réseaux de neurones dans le cerveau. Cependant, la complexité du cerveau nécessite un niveau approprié d'analyse allant des modèles « biophysiques » aux modèles plus abstraits qui embarquent tout de même les aspects clés du fonctionnement biologique. Ainsi, les niveaux d'abstraction peuvent varier des modèles biophysiques aux modèles théoriques en passant par des circuits neuraux sans exclure les modèles dédiés aux applications et les modèles génériques (Voir Figure I.2). D'un côté, les modèles biophysiques tels que GENESIS [28] et NEURON [29] intègrent les propriétés moléculaires, électriques et morphologiques des neurones. D'un autre côté, les modèles théoriques sont développés pour comprendre des aspects cognitifs plus abstraits du fonctionnement du cerveau tels que le raisonnement et la conscience [30]. Ces modèles proposent des principes démontrés tout en mettant de côté les mécanismes biophysiques des neurones. Entre ces deux extrêmes, les modèles basés sur les circuits neuraux s'inspirent du fonctionnement des neurones avec une approche électrique en termes de transmission des signaux. Les modèles dédiés aux applications simplifient les détails de la circuiterie du cerveau afin de les utiliser dans des applications spécifiques telles que l'audition, la navigation ou encore la reconnaissance d'objets [31]. Les modèles génériques utilisent des algorithmes qui se basent sur les principes fondamentaux du système nerveux (auto-organisation, retro-propagation de l'erreur, …).

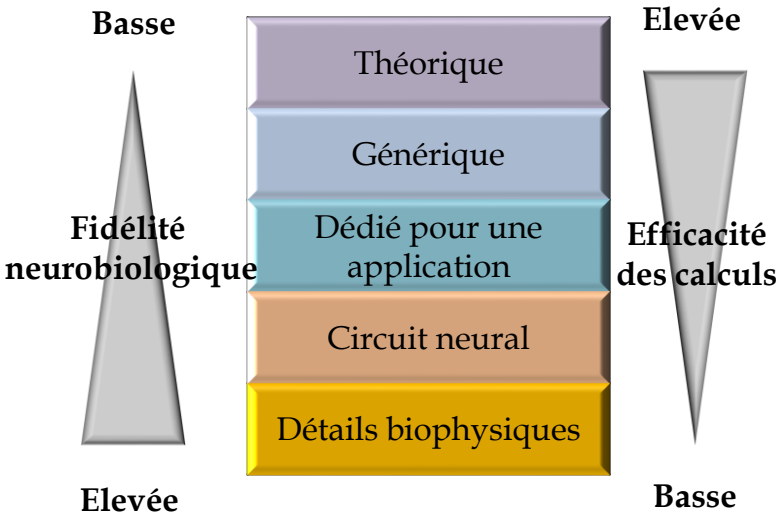


Figure I.2 Niveaux d'abstraction des modèles basés sur le cerveau. Image inspirée de [32]

B. Projets relatifs aux architectures et modèles neuro-inspirés

Il existe de nombreux projets portant sur le cerveau ainsi que sur les réseaux de neurones artificiels tels que le projet SpinNaker [33], le Human Brain Project (HBP) [1], [34], [35] ou le projet SyNAPSE [36].

Le projet SpinNaker [33], [37] est un projet débuté en 2005 et réalisé au sein de l'APT group (Advanced Processor Technologies research group) de l'université de Manchester. Il a pour principal objectif la mise en œuvre d'un ordinateur massivement parallèle muni de près d'un million de cœurs dont les caractéristiques en termes de connectivité et de calcul sont inspirées par le cerveau des mammifères. Cet « ordinateur » doit permettre la mise en œuvre de très larges réseaux de neurones à impulsion (*spiking neural network*). Les systèmes les plus récents atteignent 2500 processeurs de calcul au sein de la même plateforme. Une machine SpinNaker (Voir Figure I.3) est une grille de nœuds 2D avec chaque nœud incorporant 18 processeurs ARM968 avec 96kB de mémoire locale, 128MB de mémoire partagée et que des opérateurs entiers sur 32bits. Chaque processeur est capable d'émuler une centaine de neurones. L'ensemble est donc capable de simuler des réseaux contenant plus d'un million de neurones [38] [39].

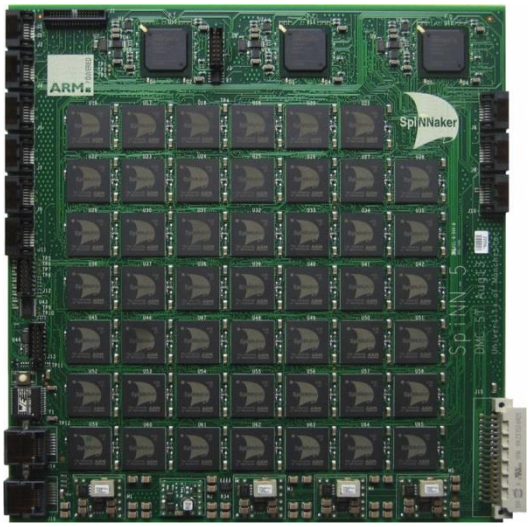


Figure I.3 Aperçu d'un système SpinNaker muni de 48 nœuds [39]

Le HBP est un projet européen débuté en 2013 pour une durée de 10 ans. Il contient en son sein une communauté de projets très variés et a pour objectif principal une meilleure représentation architecturale, structurelle et fonctionnelle du cerveau. Les sous-objectifs de ce projet sont [40] : la compréhension et la modélisation du fonctionnement du cerveau, la modélisation de maladies psychiatriques et neurologiques afin de développer des traitements, la spécification de nouvelles technologies neuromorphiques et neurorobotiques inspirées par le cerveau. Le projet SpinNaker s'insère aussi dans le cadre du HBP afin de simuler les modèles proposés dans le cadre de ce projet [33][41].

Le projet SyNAPSE d'IBM [36] est un projet financé par la DARPA (Defense Advanced Research Projects Agency). Ce projet a pour objectifs principaux d'une part la conception de puces *cognitive*s très basse consommation permettant de simuler certaines fonctionnalités du cerveau humain [42] (apprentissage et restitution d'information, classification, etc.) et d'autre part l'intégration de ces puces bio-inspirées dans des équipements tels que les ordinateurs, les robots, les smartphones, les tablettes et autres équipements électroniques (…) afin de pouvoir fabriquer des machines « intelligentes ». Ces machines seront ainsi capables de réaliser de manière efficace de nombreuses tâches telles que l'assistance aux utilisateurs, la navigation automatique, la détection et la classification multi-objets dans des images provenant d'une caméra. Les puces réalisées à travers le projet SyNAPSE se basent sur l'architecture TrueNorth qui permet de concevoir des systèmes multi-cœurs (4096 cœurs en 28nm) ayant une faible consommation énergétique (70mW en exécutant un réseau récurrent de base ce qui est quatre fois plus faible que la consommation d'un ordinateur classique manipulant le même réseau). Les différents cœurs sont interconnectés à travers un réseau NOC (Network On Chips). En plus de concevoir des puces intégrables, l'objectif est aussi de réaliser des systèmes mettant en œuvre de très larges réseaux à impulsion. En effet, bien que des systèmes manipulant 16 millions de neurones et 4 milliards de synapses aient

déjà été construits, l'objectif est maintenant de concevoir un système capable de manipuler jusqu'à 4 milliards de neurones et 1 trilliard de synapses, le tout tenant dans une étagère [36].

C. Historique des modèles de réseaux de neurones artificiels

Les réseaux de neurones artificiels peuvent être classés en deux familles selon leur type de calcul [43] :

- Les réseaux de la première famille sont basés sur les neurones de McCullots et Pits (perceptrons) [44] [45] qui possèdent une fonction d'activation.
- Les réseaux de la seconde famille utilisent des neurones à impulsion [43], [46], [47] dont la valeur varie avec le temps.

Dans ce document, seuls les réseaux de la première famille sont considérés.

Les propositions des modèles de réseaux de neurones artificiels ont débuté avec les travaux de McCullots et Pits [44] spécifiant le tout premier modèle formel et mathématique du neurone. Ce modèle a conduit à la spécification de nombreux modèles de réseaux de neurones, chacun pouvant différer en termes d'organisation, de fonctionnement et de fonctionnalités.

Le tout premier, le perceptron [48] contient un seul neurone pouvant prendre uniquement deux valeurs et étant capable d'apprendre l'association de stimuli à une valeur binaire en sortie. Malgré les critiques de Minsky et Papert [49] sur ses limitations, des travaux supplémentaires sur les réseaux munis de plusieurs neurones ont permis la proposition de modèles plus complets dont le perceptron multicouche [50]. Dans ce type de réseau, les neurones sont organisés en couches et les neurones des différentes couches sont interconnectés de manière hiérarchique. Comme autre exemple, les réseaux de Hopfield [51] contiennent des neurones totalement interconnectés. Les cartes auto-organisatrices de Kohonen [52] intègrent un aspect compétitif avec un unique neurone sélectionné pour être associé aux stimuli à l'entrée. D'un autre côté, dans les réseaux basés sur le modèle Cogent-Confabulation [53], les neurones sont répartis en groupes de neurones avec des connexions uniquement possibles entre les neurones de groupes différents. Tous ces réseaux utilisent des connexions à poids réels contrairement à d'autres comme les réseaux de Willshaw [54], [55] qui contiennent uniquement deux couches et où les connexions entre neurones sont binaires.

Certains modèles de réseaux de neurones comme les réseaux de Hopfield, les cartes auto-organisatrices de Kohonen ou encore les réseaux de Willshaw ont été proposés avec pour objectif d'apprendre l'association entre deux informations et de les restituer lorsqu'une partie des informations est fournie en entrée. Un système muni d'une telle fonctionnalité permet de concevoir des *mémoires associatives* [56].

Les mémoires associatives sont souvent appelées CAM [57] (Content Adressable Memory : Mémoire adressable par le contenu). Elles ont de nombreuses applications dont la conception de systèmes ayant besoin d'une recherche rapide et parallèle (Système de Gestion de Base de Données (SGBD), routeurs de réseaux informatiques) [58], [59], ou encore le traitement d'images [60], [61].

D'autres modèles de réseaux de neurones permettant de concevoir des mémoires associatives sont encore proposés aujourd'hui et ils améliorent les performances de l'état de l'art tout en introduisant de nouveaux concepts.

Récemment, un nouveau modèle a été proposé pour concevoir des mémoires associatives, le GBNN (Gripon Berrou Neural Network) [62]. Ce modèle a une capacité de stockage supérieure à celle des réseaux de Hopfield. Le GBNN s'inspire de nombreux modèles existants. Comme les réseaux de Willshaw, il utilise une approche où les connexions entre neurones sont binaires. Comme les réseaux inspirés du Cogent-Confabulation, il utilise un système de groupes de neurones et les neurones d'un groupe peuvent uniquement être connectés qu'aux neurones des autres groupes. Il est aussi très proche de ce dernier parce que l'objectif lors de la restitution d'une information est de n'avoir qu'un unique neurone sélectionné dans chaque groupe grâce à une règle de type *Winner Take All* (WTA). La propriété de n'avoir qu'un faible nombre de neurones actifs dans chaque groupe émane du principe du *codage parcimonieux* de l'information. L'usage de connexions binaires combiné à l'usage de calculs peu complexes au sein des réseaux GBNNs permet de concevoir des architectures matérielles mettant en œuvre des mémoires associatives peu coûteuses et ayant une faible consommation énergétique [58]. Dans le modèle GBNN, ces groupes de neurones sont appelés *clusters*. Les réseaux basés sur ce principe de groupes de neurones sont des *réseaux à clusters*.

Bien qu'ils aient une capacité et une efficacité supérieure (quantité maximale d'information apprises en bits divisée par le coût du stockage des connexions du réseau) par rapport aux réseaux de Hopfield (0,14 contre 0,69 pour le GBNN), les réseaux GBNN ont une performance optimale uniquement si les informations apprises ont une distribution uniforme. Des méthodes améliorant leurs performances pour des distributions non uniformes [63] ainsi que des architectures matérielles mettant en œuvre les réseaux GBNN [64], [65] ont été proposées. Cependant, d'un côté, ces solutions restent très coûteuses en

ressources matérielles. D'un autre côté, les architectures proposées sont restreintes à des réseaux de tailles fixes et sont incapables de passer à l'échelle.

Les objectifs de cette thèse sont les suivants : (1) concevoir des modèles inspirés du modèle GBNN et plus performants que l'état de l'art, (2) proposer des architectures moins coûteuses que les solutions existantes et (3) concevoir une architecture générique configurable mettant en œuvre les modèles proposés et capable de manipuler des réseaux de tailles variables.

D. Plan

Les résultats des travaux de cette thèse sont présentés à travers cinq chapitres.

Le second chapitre se focalise sur l'état de l'art des réseaux de neurones artificiels ainsi que sur le GBNN. Il vise d'une part à présenter les principaux modèles de réseaux de neurones et de mémoires associatives. Il a d'une autre part pour objectif de résumer les travaux qui ont été réalisés sur le GBNN, modèle sur lequel se basent nos travaux.

Dans le troisième chapitre, les réseaux GBNN sont considérés en termes de performance. En effet, bien qu'ils puissent apprendre plus d'informations que les réseaux de Hopfield, les réseaux GBNN sont moins efficaces lorsque les informations à apprendre sont trop nombreuses et/ou trop similaires. Dans ce chapitre, le concept de *clone de neurone* est proposé et il permet d'améliorer les performances. Plusieurs modèles de réseaux basés sur ce concept sont présentés, chacun ayant sa propre structure et ses propres méthodes d'apprentissage tout en gardant les bases et les fonctionnalités du GBNN. Bien que différents en termes de structure, ils possèdent la même dynamique pour retrouver une information endommagée. Les performances de ces modèles sont montrées à travers des évaluations et celles-ci sont comparées aux performances des approches de l'état de l'art.

Le quatrième chapitre se focalise sur le GBNN en termes d'architectures matérielles et distingue deux types d'architectures : les architectures spécialisées qui permettent de manipuler uniquement des réseaux ayant une taille prédéfinie et les architectures génériques qui permettent de manipuler des réseaux ayant des tailles variables. Après avoir réalisé un état de l'art sur les architectures mettant en œuvre les réseaux GBNN, une étude en profondeur est réalisée sur les GBNNs en termes de calcul et de stockage de leurs poids synaptiques. En effet, les calculs réalisés dans un GBNN s'apparente à une multiplication d'une matrice parcimonieuse par un vecteur. En ce sens, il existe plusieurs modes de stockage des matrices parcimonieuses ainsi que des architectures matérielles performantes capables de réaliser ce type de calcul pour des structures (matrices et vecteurs) de tailles variables. Ainsi, après avoir présenté les principaux modes de stockage des matrices parcimonieuses, ce chapitre s'achève par la présentation d'une architecture performante

réalisant le produit de matrices parcimonieuses par des vecteurs. Cette architecture sert de base pour la conception d'architectures génériques mettant en œuvre les réseaux à clusters.

Dans le cinquième chapitre, des simplifications du modèle GBNN et des architectures matérielles y relatives sont exposées. Ces simplifications permettent d'assouplir les calculs ayant lieu au sein du modèle sans pour autant modifier ni sa fonctionnalité, ni sa performance. Les architectures matérielles proposées se basent sur la simplification du modèle ainsi que sur des techniques telles que la sérialisation des calculs et des communications. Elles permettent de mettre en œuvre des réseaux ayant une taille plus large que celles atteintes par l'état de l'art. Ce résultat est démontré par de nombreuses évaluations et comparaisons du coût des ressources matérielles requises par les architectures matérielles proposées face à celui des architectures matérielles de l'état de l'art.

Le sixième chapitre aborde la conception d'une architecture générique mettant en œuvre les modèles proposés. Après avoir proposé un mode de représentation des matrices parcimonieuses simple et plus performant que les modes classiques existants, l'architecture est présentée en termes de structure et de fonctionnement. Des améliorations de cette architecture sont aussi proposées, chacune réduisant soit le temps requis pour la restitution d'une information, soit le coût en ressources de calcul. Ces améliorations sont : la fusion des étapes de calcul pour réduire le temps de calcul et l'utilisation de l'allocation dynamique de la mémoire pour pouvoir manipuler des réseaux de plus grandes tailles tout en utilisant les mêmes ressources. Des évaluations ont été réalisées afin de montrer l'effet de chacune de ces améliorations sur le coût en ressources et sur le temps de calcul.

Ce mémoire s'achève par une conclusion et des perspectives. La conclusion résume les travaux réalisés dans le cadre de cette thèse tandis que les perspectives sont exposées pour chaque thème présenté dans les différents chapitres : modèles permettant l'amélioration des performances et architectures matérielles génériques.

CHAPITRE II. Etat de l'Art des Modèles de Réseaux de Neurones

Sommaire du Chapitre :

Dans ce chapitre, les principaux modèles de réseaux de neurones et de mémoires associatives sont présentés, ainsi que le GBNN, modèle sur lequel se focalise cette thèse.

A. Introduction

Les premiers modèles de réseaux de neurones artificiels ont découlé en partie des travaux de McCullot et Pits [44] qui ont spécifiés le premier modèle mathématique du neurone. Ce modèle se base sur la composition et le comportement du neurone biologique. Le neurone biologique est l'unité fonctionnelle de base du système nerveux. Il est composé de quatre principales parties : le *corps cellulaire*, les *dendrites*, l'*axone* et les *synapses*.

Le *corps cellulaire* est le centre de traitement de l'information provenant de l'extérieur. Celle-ci peut provenir soit de capteurs sensoriels (associés à la vue ou au toucher par exemple), soit d'autres neurones.

Les *dendrites* sont des canaux de communication qui reçoivent les informations issues de l'extérieur.

L'*axone* est un canal de communication qui transmet les informations issues du corps cellulaire vers l'extérieur.

Les *synapses* sont des jonctions entre les dendrites et l'axone.

Un neurone biologique peut donc être considéré comme un élément de calcul :

- recevant des entrées modulées par les *synapses* et transitant à travers les *dendrites* ;
- traitant l'information au sein du *corps cellulaire* ;
- et transmettant le résultat de son calcul par l'*axone*.

Notations :	
x_i	Valeur de la couche d'entrée
n_o	Neurone de la couche de sortie
w_{io}	Poids synaptique entre un neurone n_i et un neurone n_o

Le neurone formel spécifié par McCullot et Pits [44] associe des entrées à une unique sortie. Il s'agit d'un processeur de calcul n_o qui reçoit des données (x_i) qui vont être modulées par des *poids synaptiques* (w_{io}), dont la somme est contrôlée par une *fonction d'activation f.* Cette fonction produit un résultat, la *valeur d'activation* y_o qui est la sortie du système. y_o est fixée entre deux valeurs limites (0/1 ou -1/1 généralement).

La Figure II.1 montre l'aperçu d'un neurone formel.

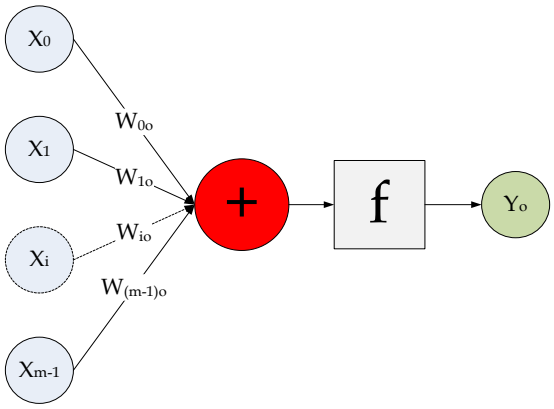


Figure II.1 Aperçu d'un neurone formel

Soit un neurone n_o muni de m entrées $(x_i)_{0\leq i\leq m-1}$. Alors $s_o(t+1)$, la somme à un instant *t+1* des entrées pondérées par leur poids est donnée par :

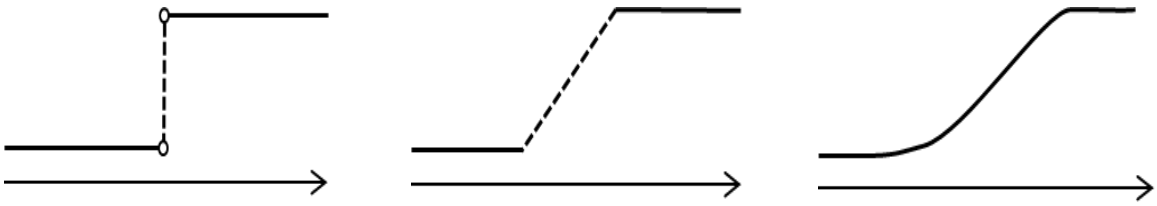
$$s_o(t+1) = \sum_{i=0}^{m-1} w_i * x_i(t) \tag{1}$$

La sortie du neurone $y_o(t+1)$ est la valeur d'activation du neurone n_o à un instant *t+1*. Elle est donnée par:

$$y_o(t+1) = f\big(s_o(t+1)\big) \tag{2}$$

Il existe plusieurs fonctions d'activation dont les plus courantes sont :

- La fonction de Heaviside (Voir Figure II.2.a)) ;
- La fonction semi-linéaire (Voir Figure II.2.b)) ;
- La fonction sigmoïde (Voir Figure II.2.c)).



$f(x) = \begin{cases} 1, si\ x < 0 \\ 0, sinon \end{cases}$

a) Heaviside

$f(x) = \begin{cases} 1, si\ x > x_2 \\ x, si\ x_1 < x < x_2 \\ 0, si\ x < x_1 \end{cases}$

b) Semi-linéaire

$f(x) = \frac{1}{1+e^{-x}}$

c) Sigmoïde

Figure II.2 Quelques exemples de fonctions d'activation

Sur le plan mathématique, un réseau de neurones artificiels est un graphe pondéré c.à.d. un ensemble de nœuds (les neurones) interconnectés par des arcs associés à des poids appelés

poids synaptiques. Sur le plan électronique et informatique, un réseau de neurones est un système distribué et parallèle composé d'éléments de calcul (les neurones) interconnectés par des liens et qui fournit un résultat à partir de données internes (les poids synaptiques) et externes.

Les paramètres d'un tel système sont [66] :

- L'ensemble des unités de calcul (les neurones) ;
- Les interconnections entre ces unités de calcul ;
- Le *type de connexions* entre les unités de calcul (binaires ou réels) et le *poids* de ces connexions ;
- La *règle de propagation* c.à.d. la manière de combiner les entrées d'une unité de calcul ;
- La *fonction d'activation* qui détermine la sortie de chaque unité de calcul à partir de la règle de propagation ;
- L'entrée externe (*biais*) de chaque unité ;
- La méthode d'acquisition de l'information (*apprentissage*).

Les réseaux de neurones sont classés selon deux principales topologies :

- Les réseaux *feed-forward* (ex : perceptron, perceptron multi-couche, …) : où les données circulent de l'entrée vers la sortie à travers les liens sans possibilité de retour en utilisant ces mêmes liens (transmission unidirectionnelle). Dans ce cas, le réseau est similaire à un graphe orienté.
- Les réseaux *récurrents* (ex : réseaux de Hopfield, machines de Boltzmann) : avec existence des retours entre l'entrée et la sortie. Dans ce cas, le réseau est similaire à un graphe non orienté.

La Figure II.3 montre les différentes classes de réseau de neurones ainsi que quelques exemples de réseaux pour chaque classe.

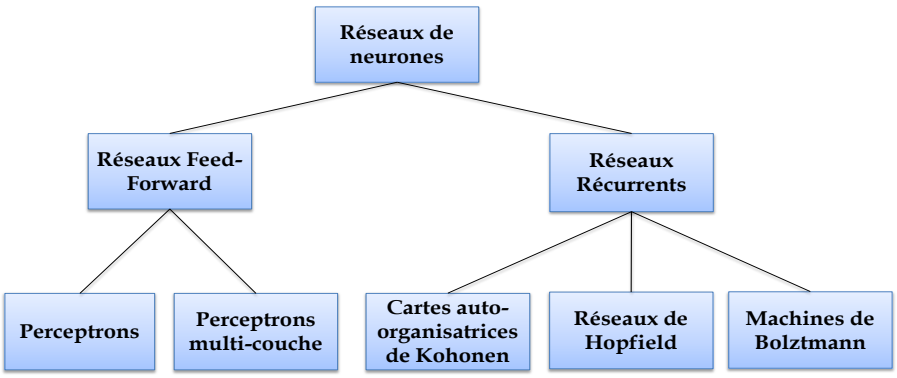


Figure II.3 Classification des réseaux de neurones selon la topologie

Il existe deux types d'apprentissage pour les réseaux de neurones artificiels :

- *Apprentissage supervisé* (ou apprentissage associatif) : le réseau est entraîné en fournissant à la fois l'entrée et la sortie du réseau. La paire entrée-sortie peut être fournie soit par un opérateur externe soit par le système lui-même (apprentissage automatique).
- *Apprentissage non-supervisé* (ou auto-organisation) : le réseau reçoit uniquement l'entrée et il évolue automatiquement de manière à reconnaître des motifs présents dans cette entrée. Le système doit alors développer sa propre représentation de ces entrées.

Les réseaux de neurones permettent concevoir des *mémoires associatives*. Il s'agit de mémoires capables d'apprendre l'association entre deux informations I_0 et I_1 et de retrouver I_1 si I_0 est fournie comme entrée. Il existe deux principaux types de mémoires associatives [67] :

- Les *mémoires auto-associatives*, dans lesquelles $I_0 = I_1$ et qui permettent de trouver I_1 à partir d'une version bruitée d'elle-même.
- Les *mémoires hétéro-associatives,* dans lesquelles $I_0 \neq I_1$ et qui permettent de trouver I_1 à partir de I_0 . S'il est possible de trouver I_0 à partir de I_1 alors la mémoire est dite *bidirectionnelle* [67], [68].

La caractéristique principale d'une mémoire associative est sa *capacité* [67]. Selon Palm, il s'agit de la quantité maximale d'informations (en bits) qu'elle peut apprendre divisée par la coût de la matrice d'adjacence. Cette notion est liée à la *diversité* [62] qui est le nombre maximal d'informations que le système peut apprendre. La diversité est définie uniquement pour des informations ayant une distribution uniforme. Ainsi pour un système de capacité Ca, de diversité Div et apprenant des informations ayant une taille de nb bits alors sa capacité est donnée par :

$$Ca = nb * Div \tag{3}$$

B. Réseaux feed-fordward

1. Perceptron

Le perceptron est un modèle de réseaux de neurones proposé par Rosenblatt [48]. Il est inspiré du modèle formel de neurone à l'exception qu'il est muni d'une méthode d'apprentissage. Sous sa forme la plus simple, un perceptron est constitué d'une couche contenant *N* entrées (*couche d'entrée*) nourrissant une unique unité qui est un neurone. Le perceptron est donc un *réseau* ayant une unique sortie (*couche de sortie*). Le neurone de sortie utilise comme fonction d'activation la fonction de Heaviside.

Dans le perceptron, le vecteur formé par les différentes entrées ne peut être associé qu'à une des deux informations possibles en sortie -1 et 1 (ou 0 et 1). Un perceptron réalise donc la classification d'un stimulus avec deux labels en sortie. Pour réaliser cette tâche, le perceptron doit d'abord apprendre deux ensembles de stimuli d'entrainement, chaque ensemble étant associé à chaque information en sortie.

L'apprentissage dans un perceptron se base sur l'*algorithme de convergence du perceptron* [69]. Pour une couple entrées-sortie, cet algorithme consiste à ajuster les poids synaptiques du réseau en fonction de l'erreur existante entre la sortie attendue et la sortie calculée. A l'initialisation du réseau, tous les poids valent 0. L'apprentissage d'un ensemble $\mathcal{E}$ de vecteurs se déroule en trois phases principales. Pour chaque vecteur de $\mathcal{E}$:

- Association de chaque vecteur $\boldsymbol{x} \in \mathcal{E}$ à la sortie $s(\boldsymbol{x})$ associée ;
- Calcul de la sortie réelle $y(\boldsymbol{x})$ associée à ce vecteur $\boldsymbol{x}$;
- Mise à jour des poids en fonction du résultat. Chaque poids w_{io} du réseau évolue de $w_{io}(t)$ à $w_{io}(t+1)$ selon la formule suivante :

$$w_{io}(t+1) = w_{io}(t) + \eta\big(s(\boldsymbol{x}) - y(\boldsymbol{x})\big)\boldsymbol{x} \tag{4}$$

Avec η la *constante d'apprentissage* qui est un paramètre arbitraire du réseau.

2. Perceptron MultiCouche (PMC)

Un *réseau feed-forward* contient plus de 2 couches : une couche d'entrée, une couche de sortie ainsi qu'une ou plusieurs couches cachées. Toutes les couches contiennent des neurones, sauf la couche d'entrée qui reçoit les données manipulées par le réseau.

Un *PMC* [50] est un réseau feed-forward dans lequel chaque neurone de la première couche couchée reçoit toutes les données de la couche d'entrée et chaque neurone d'une couche d'un niveau supérieur (différent de la couche d'entrée) est connecté à tous les neurones de cette couche. Dans un PMC, les neurones peuvent utiliser une fonction d'activation de type sigmoïde. Un PMC associe une entrée à une sortie et est capable de réaliser des tâches de classification, généralisation et de prédiction.

La Figure II.4 montre un perceptron multicouche muni de 3 couches : une couche d'entrée contenant 4 éléments, une couche cachée contenant 3 neurones recevant des données de la couche d'entrée, et une couche de sortie contenant elle aussi 3 neurones connectés à tous les neurones de la couche cachée.

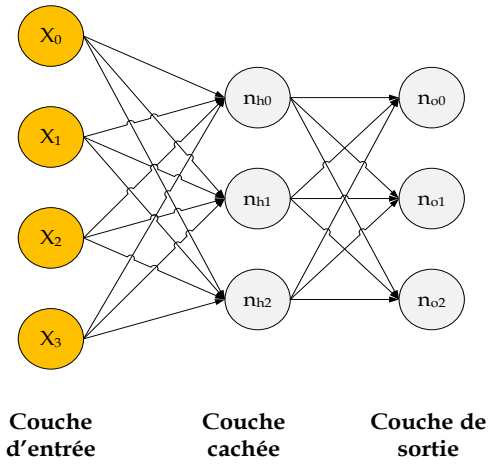


Figure II.4 Aperçu d'un perceptron multicouche

L'apprentissage dans un PMC s'effectue grâce à l'algorithme de *retro-propagation de l'erreur*.

Notations :	
$v_j^{(l)}(it)$	Valeur d'un neurone j situé dans une couche l qui n'est pas la couche d'entrée et ceci pour une itération it
$e_j^{(L)}$	Erreur entre la sortie attendue et la sortie calculée d'un neurone de sortie j situé dans la couche L
$w_{kj}^{(l+1)}$	Poids synaptique entre un neurone j de la couche $l+1$ et un neurone k de la couche l
$y_i^{(l)}$	Sortie d'un neurone n_i de la couche l

A l'initialisation du réseau, tous les poids synaptiques générés à partir d'une distribution uniforme. L'apprentissage se déroule de la manière suivante. Pour chaque échantillon de l'ensemble des vecteurs d'entraînement, les trois calculs suivants doivent être réalisés avec plusieurs itérations (it) et pour le même échantillon :

- Calcul de la valeur d'activation pour chaque neurone du réseau et pour chaque neurone de sortie, calcul de l'erreur entre la sortie attendue et la sortie calculée ;
- Calcul des gradients locaux δ_j :

$$\delta_j^{(l)}(it) = \begin{cases} e_j^{(L)}(it) f_j'\left(v_j^{(L)}(it)\right) \text{ pour chaque neurone } j \text{ de la couche de sortie } L \\ f_j'\left(v_j^{(l)}(it)\right) \sum_k \delta_k^{(l+1)}(it) w_{kj}^{(l+1)}(it) \text{ pour chaque neurone } j \text{ de la couche cachée } l \end{cases} \quad (5)$$

- Correction des valeurs des poids synaptiques :

$$w_{ji}^{(l)}(it+1) = w_{ji}^{(l)}(it) + \alpha\left[w_{ji}^{(l)}(it-1)\right] + \eta\delta_k^{(l)}(it)y_i^{(l-1)}(it) \quad (6)$$

- Evaluation de la condition d'arrêt : il existe plusieurs types de condition d'arrêt. L'une d'entre elles est la suivante : l'apprentissage pour l'entrée en cours s'arrête lorsque la variation des valeurs des poids synaptiques entre deux itérations est suffisamment faible (0,1 à 0,01%).

3. Cogent Confabulation

Le modèle *Cogent Confabulation* [53] est un modèle proposé par Hecht-Nielsen qui est basé sur les théories du fonctionnement du cortex cérébral. D'après cette théorie, le cortex cérébral humain possède environ 4000 modules disjoints. Ces modules, encore appelés *lexiques,* contiennent des *symboles*. Un symbole est une représentation mentale d'un élément du monde réel. Les symboles sont représentés physiquement par des groupes de neurones. Ces neurones peuvent uniquement être connectés à des neurones d'autres modules par des liens appelés *liens de connaissance*. Les connexions sont unidirectionnelles et ont des poids qui sont généralement non binaires. Un réseau Cogent-Confabulation est typiquement constitué d'une couche d'entrée et d'une couche de sortie, chacune contenant des lexiques. Les symboles des lexiques de la couche d'entrée sont connectés aux symboles des lexiques de la couche de sortie. Chaque entrée active un unique symbole faisant parti d'un unique lexique de la couche d'entrée.

La Figure II.5 montre un réseau de ce type avec 3 lexiques à l'entrée et 1 lexique en sortie. Chaque lexique contient des symboles qui représentent ici des lettres de l'alphabet latin. Un tel système est capable de déterminer la dernière lettre d'un mot de quatre lettres.

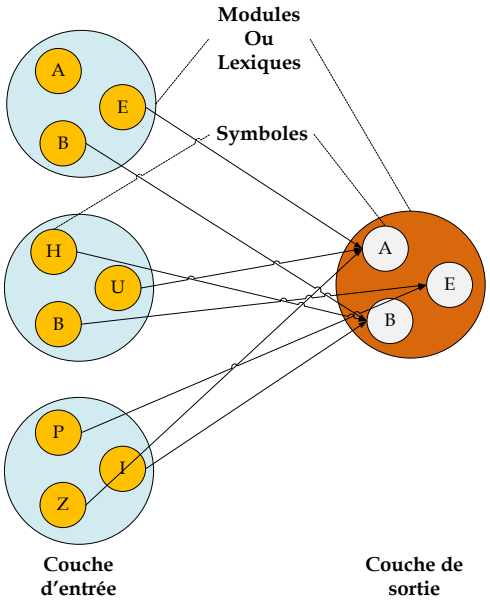


Figure II.5 Aperçu d'un réseau basé sur le principe du Cogent-Confabulation

Lorsque le système reçoit des stimuli extérieurs, certains symboles deviennent actifs. Ces symboles excitent les symboles auxquels ils sont connectés et se trouvant dans les autres modules. L'opération de *confabulation* a alors lieu. Celle-ci consiste à ne laisser actifs que les symboles ayant l'excitation la plus grande dans leur module.

L'apprentissage dans un modèle Cogent-Confabulation se réalise par application de la loi de Hebb sur les liens de connaissance entre les symboles. Pour deux symboles i et o appartenant à des module différents j et j' et connectés par un lien de connaissance, la valeur de ce lien w_{io} est donné par :

$$w_{io} = p(i|o) = \frac{N_{io}}{N_i} \tag{7}$$

Avec N_i le nombre total de fois que le symbole i a été déclenché durant l'apprentissage et N_{io} le nombre de fois où les symboles i et o ont été déclenchés en même temps durant l'apprentissage. $p(i|o)$ est la probabilité conditionnelle caractérisant l'apparition de i en fonction de o.

L'opération de *confabulation* consiste à déterminer dans le/les lexique(s) de la couche de sortie quels sont les symboles ayant la plus grande probabilité d'être déclenchés par les symboles actifs dans les modules de la couche d'entrée.

Cette opération se déroule en deux étapes :

- Pour chaque symbole dans chaque lexique en sortie, calcul de l'excitation totale due aux symboles présents à l'entrée : il s'agit d'un score qui dépend principalement de la valeur des liens de connaissance entre les symboles déclenchés dans la couche d'entrée et les symboles présents dans la couche de sortie.
- Pour chaque lexique de sortie, sélection du (ou des) symbole(s) ayant la plus grande excitation dans son module (WTA).

L'excitation $ex(o)$ d'un symbole o dans un lexique L_o est donnée par [70], [71] :

$$ex(o) = \sum_{L_i \subset L_o} \sum_{i \in L_i} I(i) \ln\left(\frac{p(o|i)}{P_o}\right) + B \tag{8}$$

Avec L_i un lexique dont les symboles ont des connexions avec les symboles de L_o. i est un symbole du lexique L_i. P_o est la plus petite valeur informative de $p(o|i)$. $I(i)$ est une fonction telle que $I(i) = 1$ si le symbole i est présent à l'entrée et 0 sinon. B est un paramètre appelé *saut de bande*. Il assure qu'un symbole ayant des connexions avec le plus de lexiques aura une excitation plus grande qu'un autre symbole du même lexique avec moins de connexions.

Ici, les valeurs d'activation sont binaires. Un symbole a une valeur d'activation 1 s'il possède l'excitation maximale dans son lexique.

C. Réseaux récurrents

1. Réseaux de Hopfield (HNN)

Les HNN [51] sont des réseaux dans lesquels tous les neurones sont interconnectés par des connexions à poids réels. Ici, il n'y a qu'une seule couche et la couche de sortie est assimilée à la couche d'entrée. De plus, chaque neurone possède une valeur d'activation binaire (0/1 ou -1/1). L'ensemble des valeurs d'activation des neurones d'un HNN représente son état et sa sortie.

La Figure II.6 montre un HNN contenant 6 neurones.

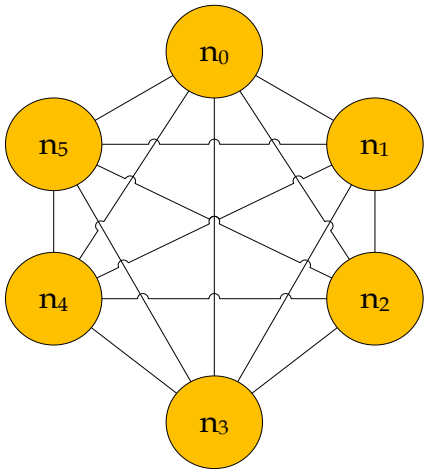


Figure II.6 Aperçu d'un réseau de Hopfield contenant 6 neurones

Notations :	
$w(n_i, n_{i'})$	Poids synaptique entre un neurone i et un neurone i'
$v_k(n_i)$	Valeur du neurone n_i pour le $k^{ième}$ vecteur d'entraînement

Soit un HNN muni de N neurones et qui doit apprendre np patterns. Soient deux neurones n_i et $n_{i'}$ $(i \neq i')$ relié par un poids $w(n_i, n_{i'})$. Soit $v(n_i)(t)$ l'état d'un neurone à un instant t. Soit $\boldsymbol{x}$ un vecteur binaire tel que $|\boldsymbol{x}| = N$. Alors, ce HNN peut apprendre le vecteur x et le restituer s'il reçoit une version de ce dernier mais contenant des bits inversés.

Si un HNN apprend np vecteurs d'entraînement, le poids $w(n_i, n_{i'})$ entre deux neurones n_i et $n_{i'}$ est calculé de la manière suivante :

$$w(n_i\,,n_{i'}\,) = \frac{1}{np} * \left(\sum_{k=0}^{np-1} v_k(n_i) * v_k(n_{i'}) \right) \quad (9)$$

Un HNN est associé à une fonction énergie $E(t)$ à un instant t. Cette fonction est définie par :

$$E(t) = -\frac{1}{2} * \sum_{i=0}^{N-1} \sum_{i'=0}^{N-1} w(n_i\,,n_{i'}\,) * v(n_i\,)(t) * v(n_{i'}\,)(t) \quad (10)$$

Après avec avoir reçu un vecteur, les valeurs des neurones peuvent changer. Un HNN se devient stable si jamais la variation de l'énergie du système entre deux instants est nulle, c.à.d. si $E(t+1) = E(t)$.

L'entrée totale d'un neurone n_i à un instant *t+1* est donnée par :

$$s_i(t+1) = \sum_{i'=0, i' \neq i}^{N-1} w(n_i\,,n_{i'}\,) * v(n_{i'}\,)(t) \quad (11)$$

La fonction d'activation utilisée dans un HNN est une fonction de Heaviside.

Un HNN peut aussi être généralisé en permettant que les valeurs d'activation de ses neurones soient continues [45] (c.à.d. que les valeurs appartiennent à un intervalle continue $[0;1]$ ou $[-1;1]$) au lieu d'être binaires (c.à.d. les valeurs peuvent prendre uniquement deux valeurs dans l'ensemble $\{0,1\}$ ou $\{-1,1\}$). Pour cela, la fonction d'activation de Heaviside est remplacée par la fonction d'activation de type sigmoïde.

2. Les machines de Boltzmann

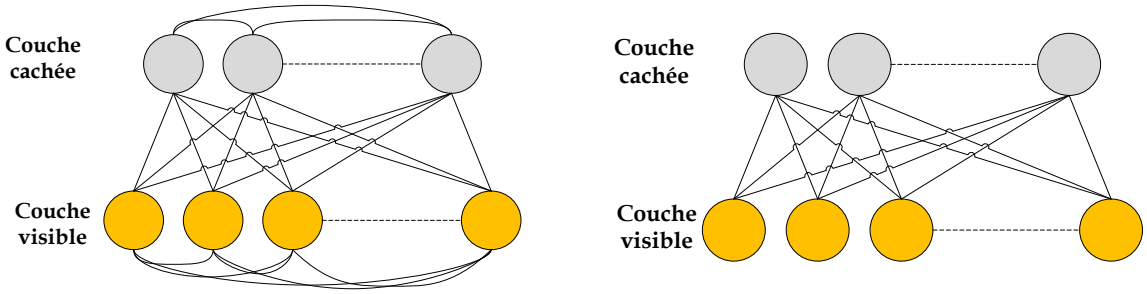
Les *machines de Boltzmann* [72]–[74] sont des réseaux de neurones où la fonction d'activation est *stochastique.* Cela signifie que dans un tel réseau, la mise à 1 de la valeur d'activation de chaque neurone dépend d'une probabilité. Les valeurs des neurones peuvent être égales à 1 ou à -1.

a) Structure

Typiquement, une machine de Boltzmann classique intègre une couche visible qui est aussi la couche de sortie, ainsi qu'une couche cachée. Les neurones de la couche visible sont connectés à tous les neurones de la couche cachée par des poids réels symétriques. Dans une machine de Boltzmann, soit tous les neurones sont connectés entre eux indépendamment de

la couche, soit les neurones d'une même couche ne sont pas connectés entre eux (connexions uniquement inter-couches). Dans ce dernier cas, la machine de Bolztmann est dite *restreinte*.

La Figure II.7 montre l'aperçu d'une machine de Boltzmann et d'une machine de Bolztmann restreinte.



a) Machine de Boltzmann b) Machine de Boltzmann restreinte

Figure II.7 Aperçu d'une machine de Boltzmann et d'une machine de Boltzmann restreinte

b) Dynamique

Pour un neurone n_i, la probabilité d'avoir une valeur d'activation à 1 est donnée par :

$$p(v(n_i) = 1) = \frac{1}{1 + e^{-s_i}} \tag{12}$$

Avec T un paramètre arbitraire encore appelé *température* du réseau.

s_i égale à :

$$s_k = \sum_{j=0, j \neq k}^{N-1} w_{kj} * x_j(t)$$

Une machine de Bolztmann est associée à une fonction énergie donnée par :

$$E(t) = -\frac{1}{2} * \sum_{k} \sum_{j,k \neq j} w_{kj} * v(n_k) * v(n_j) \tag{13}$$

Lorsque ce réseau reçoit une entrée, et si les valeurs des neurones sont calculées après avoir reçu cette entrée, celles-ci peuvent changer. Un tel changement provoque la modification de la valeur de l'énergie du réseau E. Une machine de Bolztmann atteint un état stable si et seulement si la probabilité qu'un bit change d'état devient faible :

$$P\big(v(n_i) \rightarrow -v(n_i)\big) = \frac{1}{1 + e^{\left(\frac{-\Delta E_i}{T}\right)}} \tag{14}$$

Avec ΔE_k, la variation de l'énergie si jamais, $v(n_i)$ change de valeur.

c) Apprentissage

Etant donné un ensemble de vecteurs binaires fourni au réseau, le but de l'apprentissage est de trouver les poids et les biais afin de pouvoir générer ces vecteurs avec une haute probabilité.

Il existe deux modes d'opération pour les neurones de la couche visible [69] :

- Le mode « rigide » où les neurones de la couche visible ont des valeurs spécifiques figées et déterminées par l'environnement ;
- Le mode « exécution libre » où tous les neurones peuvent se fonctionner librement.

Au début de l'apprentissage des vecteurs d'entraînement, tous les poids sont initialisés aléatoirement par des valeurs provenant d'une distribution uniforme et appartenant à $[-1; 1]$.

L'apprentissage de ces vecteurs est réalisé en deux phases :

- D'abord en mode rigide, où les valeurs des neurones de la couche d'entrée sont figées. Ainsi, pour chaque vecteur,
    - Le vecteur est fourni en entrée du réseau tandis que les neurones des couches cachées sont initialisés avec une distribution uniforme.
    - Le réseau « s'exécute » (calcul de la valeur des neurones de la couche cachée) jusqu'à atteindre un état stable.
    - Pendant le fonctionnement du réseau, la *cooccurrence* de chaque couple de neurones du réseau est calculée (probabilité que les valeurs des deux neurones soient à 1).
- Ensuite en mode libre, où les valeurs de ces neurones peuvent changer librement. Ainsi, pour chaque vecteur,
    - Le vecteur est fourni en entrée du réseau tandis que les neurones des couches cachées sont initialisés avec une distribution uniforme.
    - Le réseau « s'exécute » (calcul de la valeur de tous les neurones) jusqu'à atteindre un état stable.
    - Pendant le fonctionnement du réseau, la cooccurrence de chaque couple de neurones du réseau est calculée.
    - Ajustement de la valeur des poids en fonction des cooccurences et d'un coefficient d'apprentissage.

Soit $\rho_{jk}^{+}, j \neq k$, la corrélation entre les valeurs des neurones n_k et n_j en mode rigide et ρ_{jk}^{-}, la corrélation entre les valeurs de ces neurones en mode libre. La corrélation représente la probabilité que les valeurs de ces deux neurones soient égales. Alors, le changement Δw_{kj} appliqué au poids w_{jk} est donné par :

$$\Delta w_{jk} = \eta * (\rho_{jk}^{+} - \rho_{jk}^{-}) \quad (15)$$

Avec η le pas d'apprentissage.

3. Cartes auto-organisatrices

Dans les *cartes auto-organisatrices* (Self-Organising Maps : SOM) de Kohonen (Voir Figure II.8), le réseau est muni de deux couches. Toutes les entrées sont associées à tous les neurones de la couche de sortie. Les neurones (de sortie) sont organisés généralement selon une grille 2D. Cette organisation définit aussi pour chaque neurone quels sont ses voisins. Chaque neurone a une coordonnée précise par rapport à cette grille se situe à une distance précise des autres neurones.

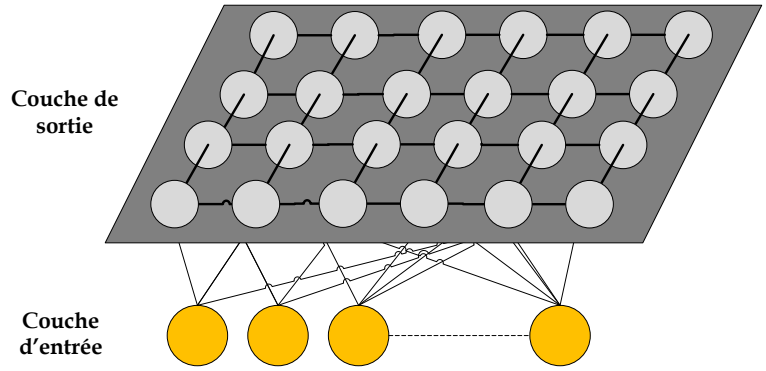


Figure II.8 Aperçu d'une carte auto-organisatrice de Kohonen

L'apprentissage dans les cartes auto-organisatrices est un *apprentissage compétitif*. Un apprentissage compétitif est une technique d'apprentissage qui consiste à élire un unique neurone qui sera associé à un stimulus présent à l'entrée du réseau. Un tel apprentissage permet de réaliser une classification des stimuli, car plusieurs stimuli peuvent correspondre à un même neurone de sortie.

Lorsque le système reçoit un vecteur à l'entrée $X = (x_i)$, le score sc_o d'un neurone de sortie n_o est défini par :

$$sc_o = \sum_i w_{io} * x_i \quad (16)$$

Le neurone de sortie qui sera sélectionné sera alors le neurone n_k ayant la valeur maximale dans tout le réseau (WTA) c.à.d. qu'il sera défini par :

$$\forall k \neq o, sc_k \geq sc_o \quad (17)$$

Une fois que le gagnant n_k a été sélectionné, alors les poids synaptiques de tous les neurones sont mis à jour. Ainsi pour un neurone n_o , si $\boldsymbol{w_o}$ est un vecteur représentant l'ensemble de ses poids synaptiques, alors W_o devient :

$$\boldsymbol{w_o}(t+1) = \boldsymbol{w_o}(t) + \gamma g(o,k)\big(\boldsymbol{x}(t) - \boldsymbol{w_o}(t)\big) \quad (18)$$

$g(o,k)$ est une fonction dont la valeur décroit au fur et à mesure que l'on s'éloigne du neurone n_k. $g(k,k) = 1$. Par exemple, celle-ci peut être égale à une gaussienne dont la valeur est maximale en k mais qui diminue lorsque l'on s'éloigne de k. Elle est alors défini par la fonction suivante :

$$g(o,k) = \exp(-(o-k)^2) \quad (19)$$

4. Réseaux de Willshaw

Un *réseau de Willshaw* [54] (Voir Figure II.9) est un réseau contenant deux couches jouant à la fois le rôle de couche d'entrée et de couche de sortie (c.à.d. les entrées du réseau représentent aussi les sorties du réseau) de telle sorte que tous les neurones d'une couche sont connectés à tous les neurones de l'autre couche par des connexions binaires.

Soit un réseau de Willshaw muni de deux couches $L^{(0)}$et $L^{(1)}$ avec chaque couche contenant N neurones. $L^{(0)}$ contient les neurones $\left(n_i^{(0)}\right)$ et $L^{(1)}$, les neurones $\left(n_{i'}^{(1)}\right)$. Lors de l'apprentissage, le système reçoit deux vecteurs binaires $\boldsymbol{v}^{(0)}$ et $\boldsymbol{v}^{(1)}$ de taille N, $\boldsymbol{v}^{(0)}$alimentant les neurones de $L^{(0)}$et $\boldsymbol{v}^{(1)}$ alimentant les neurones de $L^{(1)}$. Un réseau de Willshaw est capable d'apprendre l'association entre $\boldsymbol{v}^{(0)}$ et $\boldsymbol{v}^{(1)}$, et de pouvoir restituer $\boldsymbol{v}^{(0)}$ à partir de $\boldsymbol{v}^{(1)}$ (et vice-versa) si leur contenu a été altéré (bits inversés).

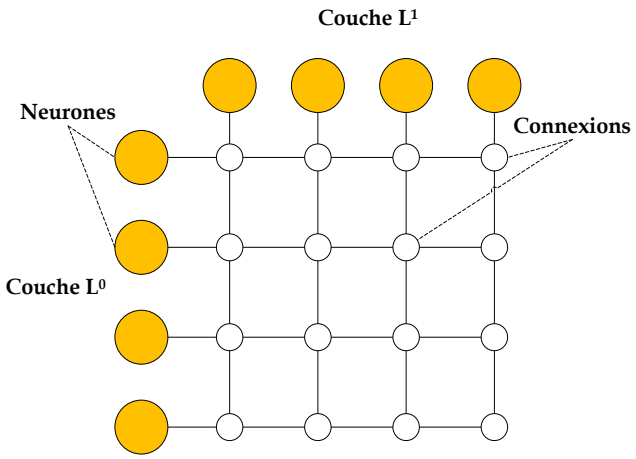


Figure II.9 Réseau de Willshaw contenant 4 neurones dans chaque couche

Lorsque le système reçoit $\boldsymbol{v^{(0)}} = [v_0^{(0)}, \dots v_i^{(0)}, \dots v_{N-1}^{(0)}]$ et $\boldsymbol{v^{(1)}} = [v_0^{(1)}, \dots v_i^{(1)}, \dots v_{N-1}^{(1)}]$, chaque neurone va prendre la valeur d'activation de son bit associé (c.à.d. $\boldsymbol{v}_i^{(0)} = v\left(n_i^{(0)}\right)$ et

$\boldsymbol{v}_{i'}^{(1)} = v\left(n_{i'}^{(1)}\right)$). L'apprentissage consiste à mettre à 1 les poids des connexions entre les neurones ayant la valeur 1 même s'ils sont déjà à 1, respectant ainsi la loi de Hebb [75].

Afin de retrouver le vecteur $\boldsymbol{v}^{(1)}$ lorsque $\boldsymbol{v}^{(0)}$ est fourni, la valeur d'activation de chaque neurone de $L^{(1)}$ est calculée en deux étapes :

- Calcul du score de chaque neurone de $L^{(1)}$ à partir des valeurs des neurones de $L^{(0)}$:

$$s\left(n_i^{(1)}\right) = \sum_{i'} v\left(n_{i'}^{(0)}\right).w\left(n_i^{(0)}; n_{i'}^{(1)}\right) \tag{20}$$

- Détermination de la valeur d'activation de chaque neurone de $L^{(1)}$:

$$v\left(n_i^{(1)}\right) = \begin{cases} 1, si\ s\left(n_i^{(1)}\right) > Val \\ 0, sinon \end{cases} \tag{21}$$

Avec Val, un paramètre arbitraire.

De même, les valeurs des neurones de $L^{(0)}$ peuvent être calculées en fonction des valeurs de neurones de $L^{(1)}$ en se basant sur le même principe.

5. Réseaux à clusters

a) Définition

Définition : Réseau à clusters

> Un *réseau à clusters* est un réseau où l'ensemble des neurones est réparti en groupes de neurones nommés *clusters* de telle sorte qu'un neurone d'un cluster ne peut être connecté qu'aux neurones présents dans les autres clusters (*neurones distants* dans les *clusters distants*)..

Notations :	
$g(C, L)$	Réseau à clusters muni de C clusters et de L neurones par cluster
$\mathcal{C}(g) = \{\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_{C-1}\}$	Ensemble des clusters du réseau g
$n_{(i,j)}$	Neurone i faisant partie d'un cluster $\mathcal{C}_j$
$w\left(n_{(i,j)}, n_{(i',j')}\right)$	Poids synaptique entre deux neurones $n_{(i,j)}$ et $n_{(i',j')}$ avec $j \neq j'$
$v(n_{(i,j)})(t)$	Valeur d'activation d'un neurone $n_{(i,j)}$ à l'instant t

Définitions : GBNN

> Un GBNN (Gripon Berrou Neural Network) [62] est un réseau à clusters où les neurones sont interconnectés par des poids *binaires* $\left(\forall i,j,i',j', w\left(n_{(i,j)}, n_{(i',j')}\right) = \{0,1\}\right)$ et *symétriques* $\left(\forall i,j,i',j', w\left(n_{(i,j)}, n_{(i',j')}\right) = w\left(n_{(i',j')}, n_{(i,j)}\right)\right)$.

Un GBNN muni de *C* clusters contenant chacun *L* neurones est noté *GBNN(C,L)*.

A l'initialisation du réseau, tous les poids sont mis à 0 (*connexion inactives*) et ces poids sont ensuite mis à 1 (*connexion actives*) au cours de l'apprentissage.

La Figure II.10 montre l'aperçu d'un GBNN(3,3) muni de trois clusters (le cluster $\mathcal{C}_0$ contenant des carrés, le cluster $\mathcal{C}_1$ contenant des ronds et le cluster $\mathcal{C}_2$ contenant des triangles). Chaque cluster contient 3 neurones. Dans la figure, chaque neurone est identifié relativement à son cluster. Par exemple, le neurone rond 2 est équivalent à $n_{(2,1)}$ (c.à.d. neurone 2 du cluster $\mathcal{C}_1$). Les connexions ayant un poids à 1 sont indiquées par les traits continus reliant les neurones (*connexions actives*).

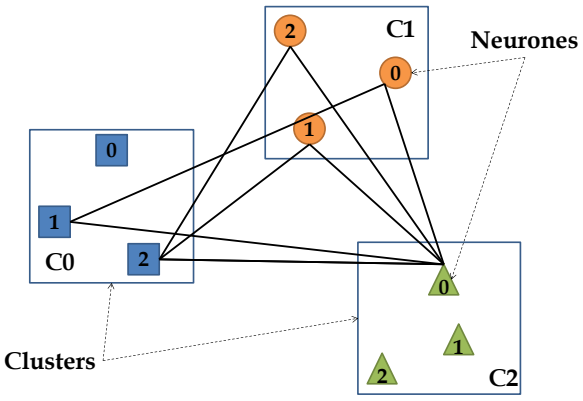


Figure II.10 Schéma d'un GBNN(3,3) muni de 3 clusters contenant chacun 3 neurones

Les poids synaptiques existants entre les neurones d'un GBNN sont stockés dans une *matrice d'adjacence symétrique*. Chaque ligne de la matrice contient les poids associés à un unique neurone et chaque cellule contient un unique poids synaptique $w\left(n_{(i,j)}, n_{(i',j')}\right)$ entre un neurone $n_{(i,j)}$ et un neurone $n_{(i',j')}$.

Le nombre total de poids synaptiques d'un GBNN(C,L) est de :

$$|\mathcal{W}| = \frac{L^2 * C * (C-1)}{2} \tag{22}$$

La Figure II.11 montre la matrice d'adjacence du GBNN(3,3) pour le réseau présenté dans la Figure II.10. Chaque sous-matrice contient les poids entre deux clusters.

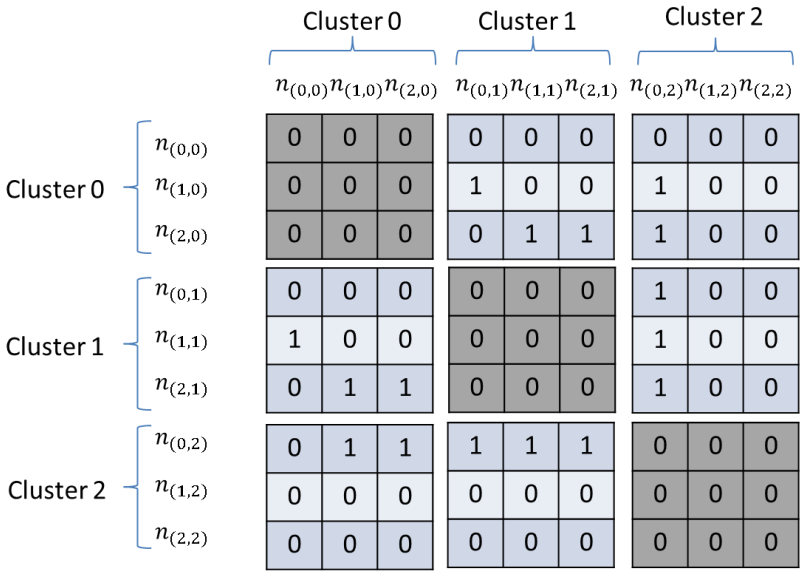


Figure II.11 Matrice d'adjacence d'un GBNN(3,3)

Les sous-matrices en gris représentent les connexions entre des neurones appartenant au même cluster. Les poids de ces connexions restent à 0 puisque dans un réseau à clusters, un neurone d'un cluster ne peut être connecté à un neurone appartenant au même cluster.

(1) Définitions

Soit $A = \{a_0, \dots a_i \dots, a_{L-1}\}$ un ensemble de L valeurs.

Soit $\boldsymbol{m} = [s_0 \dots, s_j, \dots, s_{C'-1}]$ un vecteur composé de C éléments.

Ainsi un GBNN(C,L) associé à A peut apprendre et restituer $\boldsymbol{m}$ si et seulement si le nombre de clusters est égale au nombre d'éléments de $\boldsymbol{m}$ ($C' = C$) et s'il existe au moins un élément de $\boldsymbol{m}$ qui appartient à l'ensemble A. ($\exists j / \boldsymbol{m}(j) \in A$). Dans ce cas, un unique cluster est associé à un unique symbole ($\mathcal{C}_j$ est associé à s_j) et un neurone d'un cluster est associé à la valeur prise par le symbole ($n_{(i,j)}$ est associé à s_j si $s_j = a_i$).

Dans le cadre du modèle GBNN, $\boldsymbol{m}$ est appelé *message*, s_j est appelé *symbole* et a_i est appelé *valeur de symbole*. A est appelé *alphabet*. Le terme « message » est utilisé car il fait référence au domaine des télécommunications qui est le domaine d'origine des auteurs de ce modèle [62].

Définition : Décodage

> L'opération réalisée par un GBNN et permettant de retrouver un message partiellement bruité est appelée *décodage*. Le décodage consiste à calculer l'ensemble des valeurs d'activation des neurones du réseau.

Définition : Neurone actif, Neurone inactif

Un neurone est dit *actif* si sa valeur d'activation est 1 (c.à.d. si $v(n_{(i,j)}) = 1$). Si sa valeur d'activation est 0 alors il est dit *inactif*.

Définition : Cluster connu/actif, Cluster inconnu/inactif

Un *cluster connu* ou *actif* est un cluster contenant au moins un neurone actif. Autrement, il est *inconnu* ou *inactif.*

Définition : Symbole connu, Symbole inconnu

Un *symbole connu* est un symbole dont la valeur actuelle peut être associée à un neurone du cluster associé à ce symbole. Un symbole est *inconnu* si sa valeur actuelle ne peut être associée à aucun neurone.

Définition : Alphabet

L'ensemble des valeurs que peut prendre un symbole et associé à un GBNN est appelé *alphabet.*

Par exemple, considérons un GBNN(3,3). Chaque neurone de chaque cluster peut être associé à un unique chiffre appartenant à l'alphabet $\{0, 1, 2\}$. Ainsi, ce réseau est capable d'apprendre des séquences de 3 chiffres provenant de cet alphabet.

Définition : Messages pleins, Messages parcimonieux

Un *message plein* est un message dont tous les symboles sont connus à l'apprentissage. Un *message parcimonieux* est un message qui contient des symboles inconnus. S'ils sont inconnus lors de l'apprentissage, alors ces symboles sont ignorés.

Un GBNN peut apprendre et décoder des messages pleins [62] et des messages parcimonieux [76].

Les principaux types de bruit qu'un GBNN est capable de corriger sont :

- *L'effacement* : certains symboles du message deviennent inconnus. Dans ce cas, le réseau permet de retrouver la valeur originelle de chaque symbole dont la valeur a été effacée. Par exemple, [0,0,1] devient [0,0, ?] et le troisième symbole est effacé ;
- *L'intrusion* : certains symboles du message ont leur valeur remplacée par des valeurs différentes mais connues (c.à.d. appartenant à l'alphabet). Dans ce cas, le réseau permet de retrouver la valeur originelle de chaque symbole dont la valeur a été modifiée. Par exemple, [0,0,1] devient [0,0,2] et le troisième symbole est modifié.

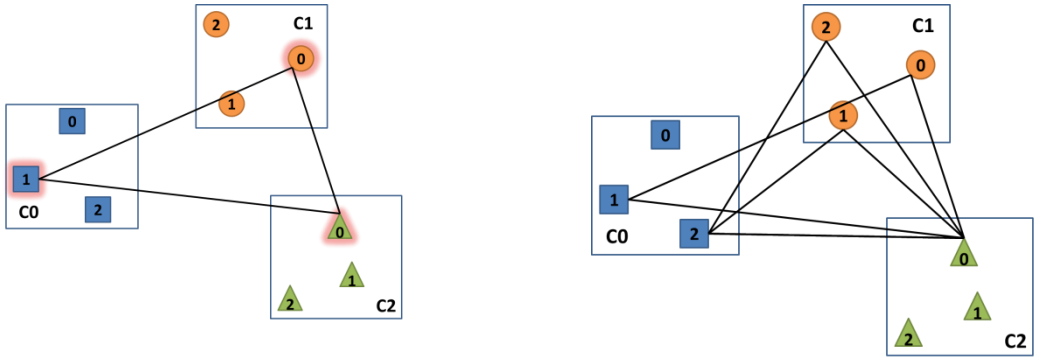
Ce document se focalise uniquement sur les messages pleins (en termes d'apprentissage) étant affecté par des bruits d'effacement (en termes de décodage) car nos travaux porte sur le modèle GBNN originel [62]. Cependant les propositions montrées dans ce document sont pertinents dans le cadre de la manipulation des autres types de message (parcimonieux) et des autres types de bruit (inversion).

Dans la suite de ce document, un symbole effacé dans un message est représenté par « ? ». Par exemple, un message [0,0,1] est considéré et si son premier symbole est effacé, alors celui-ci devient [?,0,1].

(2) Apprentissage

Lorsqu'un GBNN reçoit un message $\boldsymbol{m}$, celui-ci va provoquer l'activation des neurones qui correspondent aux valeurs de ces symboles. Ainsi, l'apprentissage consiste à mettre à 1 (activation) les connexions entre les neurones actifs même si ceux-ci sont déjà à 1 mettant ainsi en œuvre la loi d'Hebb [75]. L'ensemble formé par les neurones activés par un message et les connexions entre ces neurones définit une *clique neurale* [77]. Ainsi, d'autres messages, réutilisant ou non les mêmes connexions, peuvent être appris de la même manière.

La Figure II.12 a) montre un GBNN(3,3) ayant appris le message [1,0,0] composé de 3 symboles prenant chacun leur valeur dans l'alphabet $A = \{0,1,2\}$. La Figure II.12 b) montre le même GBNN mais ayant appris les messages [1,0,0], [2,2,0], et [2,1,0].



a) GBNN ayant appris [1,0,0]

b) GBNN ayant appris plusieurs messages

Figure II.12 Apprentissage au sein d'un GBNN(3,3)

(3) Décodage

Le décodage est l'opération permettant de retrouver un message à partir de sa version partiellement bruitée (modification de la valeur d'un sous-ensemble de symboles de ce message).

Un GBNN a réussi à restituer un message partiellement effacé si et seulement si à la fin du décodage, chaque cluster possède un unique neurone actif. C.à.d. si $\forall j, \exists! \, i, v\left(n_{(i,j)}\right)(t) = 1$.

Lorsqu'un GBNN reçoit un message à décoder, chaque symbole connu va provoquer l'activation d'un unique neurone en fonction de sa valeur. Un décodage réussi peut nécessiter une à plusieurs itérations et chaque itération consiste à déterminer la valeur d'activation de chaque neurone du réseau.

Pour une itération *t*, la détermination de la valeur de chaque neurone est réalisée en deux étapes :

- Calcul du score de chaque neurone du réseau (*Sum of Sum : SOS*).

$$sc\left(n_{(i,j)}\right)(t+1) = \gamma * v\left(n_{(i,j)}\right)(t) + \sum_{j'=0, j' \neq j}^{C-1} \left(\sum_{i'=0}^{L-1} \left(w\left(n_{(i,j)}; n_{(i',j')}\right) * v\left(n_{(i',j')}\right)(t) \right) \right) \quad (23)$$

- Calcul de la valeur d'activation de chaque neurone: la valeur d'un neurone est mise à 1 s'il a un score égal au score maximal du réseau (WTA), sinon elle est mise à 0 :

$$v\left(n_{(i,j)}\right)(t+1) = \begin{cases} 1, si\ sc\left(n_{(i,j)}\right)(t+1) = max\left(\left(sc\left(n_{(i,j)}\right)(t+1)\right)_{0 \leq i \leq L-1}\right) \\ 0, sinon \end{cases} \quad (24)$$

Un autre formule de calcul de score a été proposée dans [78] et permet d'améliorer la performance. Dans cette proposition, le calcul de score majore la *contribution* de chaque cluster $\left(\sum_{i'=0}^{L-1}\left(w\left(n_{(i,j)}; n_{(i',j')}\right) * v\left(n_{(i',j')}\right)(t)\right)\right)$ à 1. Ainsi, le score est remplacé par la somme des maxima (*Sum Of Max : SOM*) :

$$sc\left(n_{(i,j)}\right)(t+1) = \gamma * v\left(n_{i,j}\right)(t) + \sum_{j'=0, j' \neq j}^{C-1} \left(\max\left(\left(\sum_{i'=0}^{L-1} \left(w\left(n_{(i,j)}; n_{(i',j')}\right) * v\left(n_{(i',j')}\right)(t) \right) \right); 1 \right) \right) \quad (25)$$

Toutes ces méthodes permettent de corriger des bruits de type inversion ainsi que des messages parcimonieux affectés par des bruits de type effacement et/ou inversion.

b) Caractéristiques et Performances du GBNN

Les caractéristiques d'une mémoire associative à poids binaires sont principalement :

- Sa *densité d* d'un neurone ou d'un réseau c.à.d. le pourcentage de ses poids mis à 1.
- Sa *diversité Div,* c.à.d. le nombre maximal de messages qu'elle peut apprendre.

La *densité locale* fait référence à la densité d'un neurone par opposition à la *densité globale* qui fait référence à un réseau.

La diversité d'un GBNN(C,L) est donnée par [62] :

$$Div = \frac{(C-1) * N^2}{2 * C^2 \log_2\left(\frac{N}{C}\right)} \quad (26)$$

Avec $N = C * L$, le nombre total de neurones dans le réseau.

La performance du GBNN est évaluée par la *probabilité d'erreur*. Celle-ci est approximé par le *taux d'erreur de décodage* (TED) qui est égal au nombre de décodages échoués par rapport au nombre total de décodages. La performance peut aussi être évaluée par le nombre de messages appris correspondant à un TED donné.

Le TED d'un GBNN dépend d'une part du nombre de messages appris et d'autre part de la distribution de ces messages [63]. Pour un même nombre de messages appris, un GBNN possède un TED plus faible lorsque ces messages ont une distribution uniforme en comparaison du cas où ceux-ci ont une distribution non-uniforme.

La Figure II.13 montre le TED pour deux GBNN(8,32). L'un a appris des messages ayant distribution uniforme (courbe foncée) tandis que l'autre a appris des messages ayant une distribution non uniforme (courbe claire). Pour 500 messages appris, le TED d'un GBNN(8,32) est de 30% lorsque la distribution des messages appris est uniforme tandis qu'il est de 94% lorsque celle-ci est gaussienne (moyenne 16, écart-type 5).

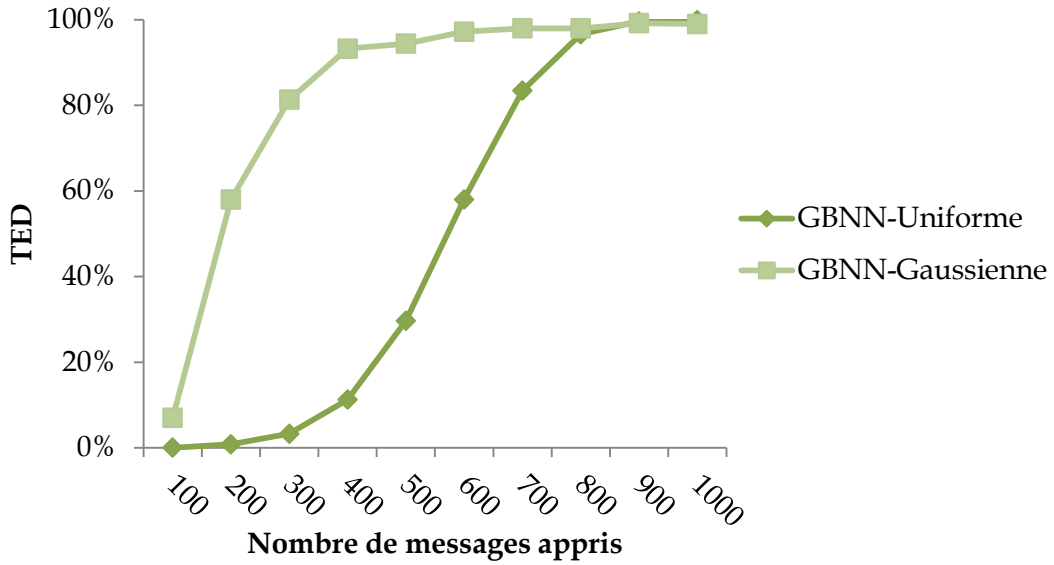


Figure II.13 Performance de deux GBNNs pour différentes distributions de messages appris

D. Bilan

Dans ce chapitre, des modèles de réseaux de neurones artificiels ont été présentés ainsi que le GBNN, modèle sur lequel se sont focalisés nos travaux. Cette étude montre que les

réseaux GBNN sont peu complexes car ils utilisent des poids binaires ainsi que des opérations simples pour restituer des informations partiellement bruitées.

Bien qu'ils aient une capacité supérieure aux HNNs, leur performance varie en fonction des messages appris ce qui nécessite de trouver des solutions pour améliorer leur performance. Des méthodes ont été proposées pour améliorer leurs performances [63], mais elles sont coûteuses en termes de ressources de stockage. Dans le prochain chapitre, ces méthodes sont présentées et le concept de *clone* ainsi que de nouveaux modèles de réseaux de neurones, les *réseaux à clones* sont proposés. Ces modèles possèdent de meilleures performances que l'état de l'art tout en conservant les mêmes propriétés (en termes d'usage de clusters, de poids binaires et de calculs simples) et les mêmes fonctionnalités (en termes d'apprentissage et décodage de messages) que le modèle GBNN originel.

CHAPITRE III. Clones et Réseaux à clones

Sommaire du Chapitre :

Dans ce chapitre, le concept de clone de neurones et des modèles de réseaux de neurones basés sur concept sont présentés. Ces modèles permettent d'obtenir de meilleures performances de décodage lorsque la distribution des messages à apprendre est non-uniforme.

A. Introduction

La performance d'un GBNN diminue d'une part en fonction de sa densité et d'autre part en fonction de la *distribution* des messages appris. Ce problème de performance peut être visualisé en observant l'évolution de la densité locale de ses neurones ainsi que celle de sa densité globale, par rapport à la distribution des messages appris.

Définition : Probabilité d'apparition de la valeur d'un symbole

Soit $\mathcal{E}$, un ensemble de B messages tel que $\mathcal{E} = \{\boldsymbol{m_q}, 0 \leq q \leq B-1\}$. Soit $A = \{a_0, \dots, a_{L-1}\}$ un alphabet de L valeurs. Chaque message $\boldsymbol{m_q} = [s_0, \dots, s_{C-1}]$ de $\mathcal{E}$ contient C symboles s_j tels que $s_j \in A$. Soit $N_j(a_i) = \{\boldsymbol{m_q},\ \boldsymbol{m_q} \in \mathcal{E}\ et\ s_j = a_i\}$, l'ensemble des occurrences où le symbole s_j prend la valeur a_i au sein des messages dans $\mathcal{E}$. Alors la *probabilité d'apparition* $F_j(a_i)$ *de la valeur* a_i *pour un symbole* s_j est donné par :

$$P_j(a_i) = \frac{\left|N_j(a_i)\right|}{|\mathcal{E}|} \tag{27}$$

Définition : Distribution

La *distribution* d'un ensemble de messages $\mathcal{E}$ est l'ensemble $Dist(\mathcal{E})$ tel que :

$$Dist(\mathcal{E}) = \{P_j(a_i), 0 \leq j \leq C-1, 0 \leq i \leq L-1\ et\ a_i \in A\} \tag{28}$$

Cela signifie que la distribution d'un ensemble de messages est l'ensemble des probabilités d'apparition des valeurs prises par les symboles contenus dans ces messages.

Définition : Distribution uniforme, Distribution non-uniforme

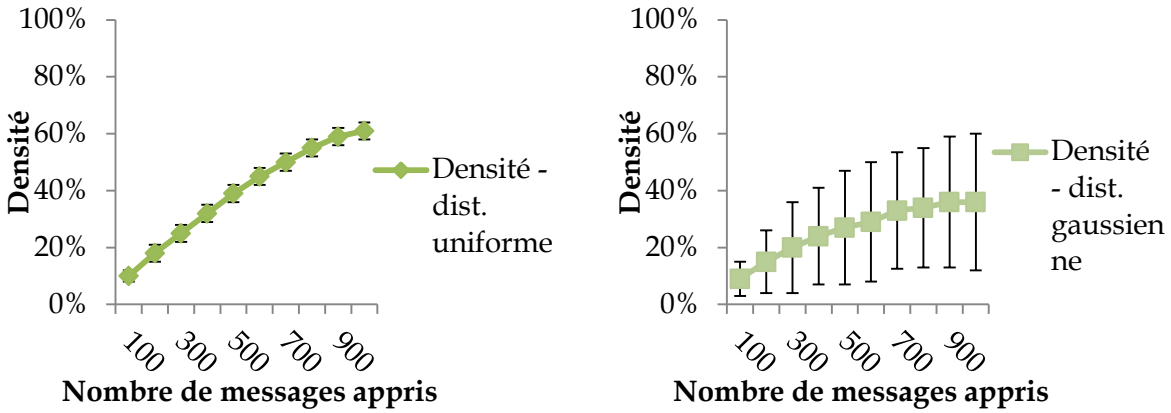
Une distribution $Dist(E)$ est dite *uniforme* si $P_j(a_i) = P_{j'}(a_{i'}) = \frac{1}{|A|}$, $0 \leq j \leq C-1$, $0 \leq j' \leq C-1, \forall a_i, a_{i'} \in A$, c.à.d. une distribution où toutes les probabilités d'apparition sont égales. Dans le cas contraire, la distribution est dite *non-uniforme*.

En pratique, une distribution de messages n'est jamais uniforme. Elle peut être proche de ce type de distribution avec une très faible variation entre les probabilités d'apparition.

Dans le cas d'une distribution uniforme, la variation entre les densités locales des neurones d'un cluster est faible comparée à la variation obtenue dans le cas d'une distribution quelconque. Dans le premier cas, les neurones sont uniformément. Dans le second cas, certains neurones sont nettement plus utilisés que les autres. Ainsi, dans le cas uniforme, les

densités locales sont proches de la densité globale tandis que dans le cas non-uniforme, les valeurs sont très différentes.

La Figure III.1 illustre les densités globales (les courbes) pour deux GBNN(8,32), l'un ayant appris des messages ayant une distribution uniforme (Voir Figure III.1.a)) et l'autre ayant appris des messages ayant une distribution gaussienne de moyenne 16 et d'écart-type 5 (Voir Figure III.1.b). Ces figures montrent aussi l'écart-type des densités locales des neurones au sein d'un cluster.



a) Cas de la distribution uniforme

b) Cas de la distribution gaussienne

Figure III.1 Densités globales de deux GBNNs (les courbes) munies de l'écart-type entre leurs densités locales (les écarts).

B. Solutions de l'état de l'art

Trois solutions ont été proposées dans [63] pour améliorer les performances du GBNN lorsque les messages à apprendre ont une distribution non-uniforme. Toutes ces propositions considèrent les messages comme étant des vecteurs binaires. Ainsi, chaque symbole est un sous-vecteur binaire :

- La première approche, nommée *ajout de clusters*, consiste à ajouter des symboles supplémentaires à chaque message à apprendre. Ainsi, cela permet au réseau de discriminer des messages similaires lors du décodage. Les bits supplémentaires appartenant aux symboles ajoutés sont générés aléatoirement à partir d'une distribution uniforme. Le nouveau message est donc appris par un réseau plus grand en termes de nombre de clusters.
- La seconde approche, nommée *ajout de bits,* consiste à ajouter des bits directement aux symboles. La séquence de bits ajoutée peut soit (1) être générée aléatoirement, soit (2) être la séquence la moins récemment utilisée. Etant donné que les symboles ont une taille plus grande, le nombre de valeurs possibles pour chaque symbole augmente. Ainsi, le nouveau message est appris par un réseau plus grand en termes de nombre de neurones par cluster.

- La troisième approche, nommée *huffman,* consiste à réaliser un codage d'Huffman [79] des valeurs de symbole en fonction de leur probabilité d'apparition. Un dictionnaire est créé à partir des messages à apprendre. Celui-ci permet d'attribuer un code à chaque valeur de symbole. Après génération des codes, les messages sont transcodés en fonction de ceux-ci. Etant donné que les codes ont des tailles différentes, celle-ci est uniformisée afin que les messages aient la même taille. L'uniformisation est réalisée en utilisant la taille du code le plus long et les bits ajoutés pour uniformiser la taille sont générés aléatoirement. Le code le plus long a une taille supérieure aux symboles de départ, ce qui augmente le nombre de codes possibles. Ainsi, comme dans le cas de l'approche *ajout de bits*, le nombre de valeurs de symboles possible augmente. Les nouveaux messages sont alors appris par un réseau plus grand en termes de nombre de neurones.

Parmi les trois solutions proposées dans [63], l'approche *ajout de bits* et l'approche *huffman* possèdent les meilleures performances et ce, pour le même coût de mémorisation (équivalent au multiple du coût de la matrice d'un GBNN : 16 GBNNs). Tandis que l'approche *huffman* a la meilleure performance mais nécessite de connaître l'ensemble des messages (*apprentissage hors-ligne*) ou alors la distribution des messages à apprendre, l'approche *ajout de bits* n'est pas soumis à cette contrainte mais possède en contrepartie une performance nettement inférieure. Des résultats chiffrés sont fournis à la fin de ce chapitre à titre de comparaison avec les solutions proposées.

C. Principe du clonage et des réseaux à clones

Afin de résoudre le problème de performance du GBNN pour des distributions non-uniformes sans pour autant avoir recours à une phase d'apprentissage hors-ligne, nous proposons de *cloner* les neurones dans le réseau. Des instances multiples d'un même neurone sont donc associées à la valeur du symbole liée à ce neurone. Ainsi, la densité locale d'un neurone sera répartie entre ses instances qui auront chacune une densité locale plus faible que celle qu'aurait dû avoir ce neurone dans le cas d'un GBNN.

Il se pose alors les deux problèmes suivants :

- Définir des modèles basés sur le concept de « clone » ainsi que la politique d'allocation (affectation) des clones aux neurones ;
- Définir des stratégies d'apprentissage pour chacun de ces modèles (C.à.d. des politiques de sélection de clones) ainsi que des méthodes de décodage.

L'objectif final est ainsi de trouver parmi tous ces modèles, celui/ceux ayant la meilleure performance. La performance d'un réseau est définit ici par le nombre de messages appris correspondant un TED donné et par rapport à un coût mémoire précis.

1. Définitions

Définition : Clone

Un *clone* est une instance d'un neurone. Un neurone peut avoir un ou plusieurs clones.

Définition : Partition

Une *partition* est un sous-ensemble de clones d'un même neurone. Un neurone peut avoir une ou plusieurs partitions.

Définition : Sous-réseau

Un *sous-réseau* regroupe une partition de chaque neurone dans chaque cluster. Chaque neurone possède exactement une partition dans chaque sous-réseau.

Définition : Réseau à clones

Un *réseau à clones* (Clone-Based Neural Network : CBNN) est composé de un ou plusieurs sous-réseaux.

La Figure III.2 donne l'aperçu d'un réseau à clones contenant 4 sous-réseaux. Chaque sous-réseau possède 3 clusters et chaque cluster intègre 3 partitions avec une partition pour chaque neurone. Chaque partition renferme 3 clones. Ainsi, chaque cluster comporte au total 3*3=9 clones (3 partitions * 3 clones). Chaque neurone est associé à 4*3=12 clones (4 sous-réseaux * 3 clones) au total.

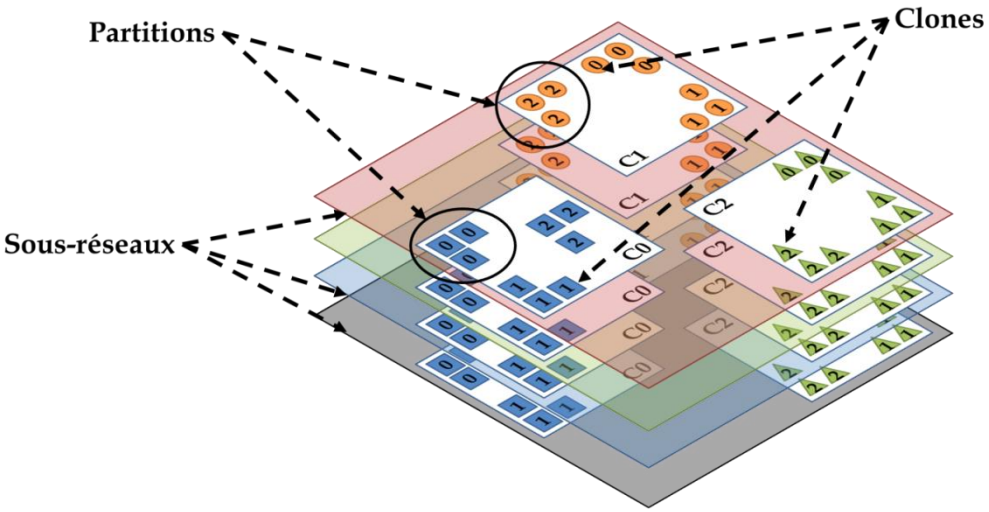


Figure III.2 Aperçu d'un réseau à clones contenant 4 sous-réseaux, 3 clusters par sous-réseaux et 3 partitions par cluster pour chaque neurone

Notations :	
$\mathcal{R}_K$	Réseau à clones muni de K sous-réseaux
R_k	Sous-réseau k de $\mathcal{R}_K$ avec $0 \leq k < K$
$\mathcal{C}(R_k) = \{\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_{C-1}\}$	Ensemble des clusters du sous-réseau R_k
$\mathcal{C}_{(j,k)}$	Cluster j du sous-réseau k
$n^x_{(i,j,k)}$	Clone x d'un neurone $n_{(i,j)}$tel que $n^x_{(i,j,k)} \in \mathcal{C}_{(j,k)}$
$w\left(n^x_{(i,j,k)};\ n^{x'}_{(i',j',k)}\right)$	Poids synaptique entre deux clones, $n^x_{(i,j,k)}$ et $n^{x'}_{(i',j',k)}$
$N_{(j,k)}$	Ensemble des clones du cluster $\mathcal{C}_{(j,k)}$
$\Gamma\left(n_{(i,j)}\right)$	Ensemble des partitions du neurone $n_{(i,j)}$

2. Propriétés

- Si un réseau à clones possède K sous-réseaux, alors $\left|\Gamma\left(n_{(i,j)}\right)\right| = K, \forall i, j$. C.à.d. qu'un neurone possède autant de partitions qu'il existe de sous-réseaux et que tous les neurones ont toujours le même nombre de partitions.
- Si $p \subset \Gamma\left(n_{(i,j)}\right)$ alors $|p| \geq 0$. C.à.d. qu'une partition d'un neurone peut être vide et ne contenir aucun clone.
- $w\left(n^x_{(i,j,k)};\ n^{x'}_{(i',j',k')}\right) = \begin{cases} 1 \text{ ou } 0 \text{ si } k = k' \text{ et } j \neq j' \\ 0, \text{sinon} \end{cases}$. C.à.d. que deux clones ne peuvent être connectés que s'ils appartiennent au même sous-réseau ($k = k'$) tout en appartenant à des clusters différents ($j \neq j'$).
- Un sous-réseau R_k est une entité indépendante capable d'apprendre et de restituer des messages.
- Si un réseau à clones possède K sous-réseaux alors un neurone $n_{(i,j)}$ est associé à K valeurs dont une valeur $v(n_{(i,j)}; k)$ pour chaque sous-réseau R_k. $v\left(n_{(i,j)}; k\right)$ est donc la valeur du neurone $n_{(i,j)}$ selon le sous-réseau k.

3. Classification des réseaux à clones

Les réseaux à clones sont divisés en deux principales classes : les *réseaux à allocation statique de clones* et les *réseaux à allocation dynamique de clones*.

Définitions : Réseau à allocation statique de clones

Un *réseau à allocation statique de clones* est un réseau dont le nombre de total de clones alloués à chaque neurone reste constant. Dans un tel réseau, le nombre de partitions (de sous-réseaux) disponibles et le nombre de clones alloués à chaque partition ne peuvent pas changer et dépendent uniquement de l'allocation initiale.

Définitions : Réseau à allocation dynamique de clones

Un *réseau à allocation dynamique de clones* est un réseau dont le nombre total de clones alloués à chaque neurone peut être augmenté au cours de l'apprentissage. Dans un

tel réseau, le nombre de partitions (de sous-réseaux) disponibles et/ou le nombre de clones alloués à chaque partition peuvent augmenter.

Dans le reste du document, les différentes variantes de réseau à clones sont nommées en utilisant 4 lettres AXBY. Chaque lettre peut prendre uniquement deux valeurs possibles :

- A indique si le *nombre de partitions alloué par neurone* (et donc de sous-réseaux) est Statique ('S') ou Dynamique ('D') ;
- X indique si le *nombre de partitions par neurone* (et donc de sous-réseaux) est un ('1') ou multiple ('N') ;
- B indique si le *nombre de clones alloué à chaque partition* est Statique ('S') ou Dynamique ('D') ;
- Y indique si le *nombre de clones alloué à chaque partition* est un ('1') ou multiple ('M').

Par exemple**,** S1SM indique qu'une unique partition est allouée par neurone (**S1**SM, un unique sous-réseau est présent dans le réseau) et qu'un nombre constant de clones (≥ 1) est alloué par partition (S1**SM**, un nombre constant de clones est associé à chaque neurone).

Comme autre exemple**,** DNDM indique que le nombre partitions par neurone et le nombre de clones par partition peuvent augmenter (c.à.d. que le nombre de sous-réseaux peut augmenter ainsi que le nombre de clones dans les partitions).

Le Tableau 1 résume les différentes variantes de réseaux à clones. Certaines combinaisons sont impossibles et ne sont pas indiquées dans ce tableau parce qu'elles sont incohérentes. Par exemple, S1D1 car le nombre de clones par partition est dynamique (S1**D**1, c.à.d. que ce nombre peut augmenter) alors qu'il est limité à 1 (S1D**1**).

Tableau 1 Variantes de réseaux à clones

Nombre de partitions par neurone / **Nombre de clones par partition**		**1**	**N**	
			Allocation Statique	**Allocation Dynamique**
1		S1S1=GBNN	SNS1	DNS1
M	**Allocation Statique**	S1SM_E ⇔ *ajout de bits* [63] S1SM_I ⇔*huffman* [63]	SNSM (_E ou _I)	DNSM
	Allocation Dynamique	S1DM	SNDM	DNDM

Dans ce même tableau, les réseaux munis du suffixe « _E » (resp. « _I ») représente des réseaux où les partitions des neurones peuvent contenir un nombre constant de clones et où le nombre de clones alloués à chaque neurone est Egal (resp. Inégal).

Les travaux présentés dans ce chapitre généralisent les modèles proposés par l'état de l'art (S1S1 et S1SM) et définissent de nouveaux modèles de réseaux à clones : SNS1, DNS1, SNSM, DNSM, S1DM, SNDM et DNDM.

4. Principe de base de l'apprentissage et du décodage

a) Apprentissage

De manière générale, l'apprentissage d'un message dans un réseau à clones se déroule en quatre phases :

- *Activation des neurones* (et donc de tous leurs clones) associés aux symboles connus du message à apprendre ;
- *Sélection d'un unique sous-réseau* (puisqu'un neurone peut avoir plusieurs clones répartis dans plusieurs sous-réseaux) ;
- *Sélection d'un unique clone actif par cluster* (puisqu'un cluster d'un sous-réseau peut contenir plusieurs clones pour un même neurone) ;
- Mise à 1 des poids entre les clones choisis.

La sélection d'un sous-réseau ou d'un clone peut se faire soit aléatoirement, soit par compétition. La compétition entre clones (resp. sous-réseaux) consiste à calculer le score des clones (resp. sous-réseaux) et à choisir le clone (resp. sous-réseau) ayant le meilleur score.

Afin d'obtenir la meilleure performance, la compétition doit prendre en compte à la fois la densité des clones actifs ainsi que les poids activés par le message à apprendre. En effet, l'apprentissage d'un message provoque l'augmentation de la densité locale des clones choisis (et donc l'augmentation de la densité globale). Cette augmentation provoque une baisse des performances.

Dans l'étude des variantes de réseaux à clones, des stratégies d'apprentissage peu complexes sur le plan calculatoire et temporel sont proposées (stratégies de sélection de sous-réseaux et stratégies de sélection de clones actifs au sein de chaque cluster du sous-réseau sélectionné). Les résultats présentés dans la section dédiée aux expériences montrent que ces stratégies sont suffisantes pour obtenir des résultats intéressants.

b) Décodage

Dans un réseau à clones, le décodage d'un message se déroule en quatre phases :

- *Activation des neurones* (et donc de tous leurs clones) associés aux symboles connus du message à décoder ;

- *Calcul du score* de tous les clones du réseau ;
- *Activation des clones* ayant un score égal au score maximal du réseau (WTA) ;
- *Calcul de l'état de tous les neurones du réseau* et selon chaque sous-réseau. Un neurone est actif dans un sous-réseau si au moins un de ses clones appartenant à ce sous-réseau est actif.

Le score $sc(n^x_{(i,j,k)})(t+1)$ d'un clone $n^x_{(i,j,k)}$ est donné par la formule suivante :

$$\boldsymbol{sc(n^x_{(i,j,k)})(t+1) = v(n^x_{(i,j,k)})(t) + \sum_{j'=0, j'\neq j}^{C-1} max\left(\sum_{n^{x'}_{(i',j',k)} \in \mathcal{N}(j',k)} \left(w\left(n^x_{(i,j,k)}; n^{x'}_{(i',j',k)}\right) * v\left(n^{x'}_{(i',j',k)}\right)(t)\right); 1\right)} \quad (29)$$

Un décodage est réussi lorsqu'à la fin du processus, le réseau ne possède qu'un unique sous-réseau avec un unique neurone actif par cluster, c.à.d. si et seulement si $\exists! k, \forall j, \exists! i, v(n_{(i,j)}; k)(t) = 1$.

Un réseau à clone est nominé en fonction de sa classification (Voir Section 3) mais aussi en fonction :

- Du nombre de sous-réseaux K ;
- Du nombre de clusters par sous-réseau C ;
- Du nombre de neurones par cluster L ;
- Du nombre total de clones Ncl présent dans chaque cluster d'un sous-réseau.

En fonction de ces paramètres, un réseau à clones pourra être noté $AXBY(K, C, L, Ncl)$. Par exemple, un réseau S1SM(1,3,4,8) possède un unique sous-réseau (1) contenant 3 clusters, chaque cluster étant muni de 8 clones, 2 clones étant alloués à chaque neurone (4*2=8).

Ainsi, sachant que dans un réseau $AXBY(K, C, L, Ncl)$ il y'a K sous-réseaux contenant C clusters ayant chacun Ncl clones, son nombre total de clones est de $Nt = K * Ncl * C$. Chaque clone étant connecté à $Ncl * (C-1)$ clones, le nombre total de poids synaptiques de ce réseau (égal au coût de mémorisation de la matrice d'adjacence en bits) est le suivant :

$$|\mathcal{W}| = \frac{K * Ncl^2 * (C-1) * C}{2} \quad (30)$$

D. Réseaux à allocation statique de clones

Un *réseau à allocation statique de clones* est un réseau dont le nombre de partitions par neurone et le nombre de clones par partition sont constants.

1. S1S1

Un réseau S1S1 contient un unique sous-réseau $(\forall i,j,|\Gamma(n_{(i,j)})|=1)$ et un unique clone est alloué dans les partitions $(\forall i,j,\forall p\in\Gamma(n_{(i,j)}),|p|=1)$. Etant munie d'un unique sous-réseau dans lequel chaque neurone ne possède qu'un unique clone, la variante S1S1 est *strictement identique* à un GBNN.

La Figure III.3 montre un S1S1(1,3,3,3) ayant un unique réseau contenant 3 clusters, chaque cluster étant muni de 3 clones, un clone étant alloué à chaque neurone.

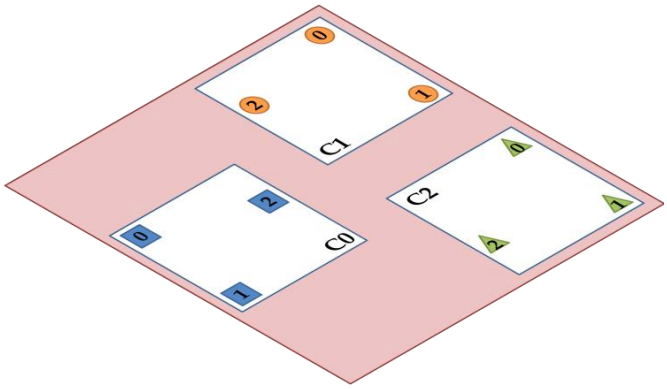


Figure III.3 Aperçu d'un S1S1(1,3,3,3)

2. S1SM

a) Définition

Un réseau S1SM contient un unique sous-réseau $(\forall i,j,|\Gamma(n_{(i,j)})|=1)$ et un nombre constant de clones est alloué dans les partitions $(\forall i,j,\forall p\in\Gamma(n_{(i,j)}),|p|=q)$.

La Figure III.4 montre un aperçu d'un S1SM(1,3,3,9) ayant un unique sous-réseau contenant 3 clusters, avec chaque cluster étant muni de 9 clones, 3 clones étant alloués à chaque neurone.

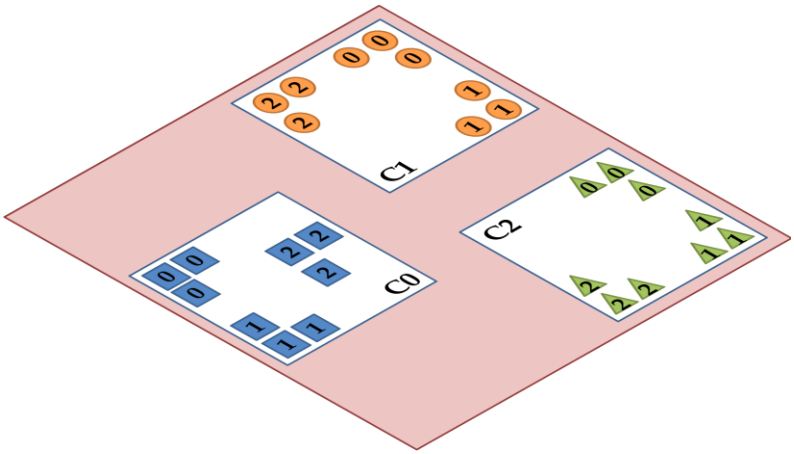


Figure III.4 Aperçu d'un S1SM(1,3,3,9)

Pour un S1SM, $K = 1$ car il ne contient qu'un unique sous-réseau. Son nombre total de poids synaptiques est donné par:

$$|\mathcal{W}| = \frac{Ncl^2 * (C-1) * C}{2} \quad (31)$$

Le nombre de clones alloués à chaque neurone peut être identique pour tous les neurones (*répartition égale*) ou différent en fonction de l'apriori sur la probabilité d'apparition des valeurs de symbole (*répartition inégale*). En fonction de cette répartition, un S1SM est noté S1SM_E (répartition Egale) ou S1SM_I (répartition Inégale).

b) Répartition égale

Dans le cas d'une répartition égale, un nombre identique de clones est alloué à chaque neurone. Un S1SM associé à ce type de partage est similaire à la deuxième approche nommée *ajout de bits* proposée dans [63]. Dans cette approche, l'ajout de bits additionnels à un symbole permet d'obtenir plusieurs valeurs de symboles ayant un même préfixe. Ces différentes valeurs générées sont donc des représentants d'une même valeur originelle et peuvent être vues comme des clones liés à cette valeur.

Notre approche par les clones présente cependant des avantages comparés à l'approche de l'état de l'art [63]. Dans [63], l'ajout de chaque bit additionnel à chaque symbole implique une multiplication par 2 du nombre de valeurs possibles (multiplication par 2 du nombre de clones) et donc une multiplication par 4 du coût de mémorisation. Dans notre approche, le nombre de clones alloué à chaque neurone ne nécessite pas d'être une puissance de 2, contrainte due à une approche purement binaire.

c) Répartition inégale

Dans le cas d'une répartition inégale, le nombre de clones alloués à un neurone dépend d'un apriori sur la probabilité d'utilisation de ce neurone (égale à la probabilité d'apparition de la valeur du symbole qui lui est associé). Cet apriori peut être obtenu par évaluation de la distribution sur un échantillon de messages provenant de l'ensemble des messages à apprendre.

Cette variante est identique à l'approche *huffman* présentée dans [63]. Dans cette approche, la taille d'un code associé à une valeur de symbole est inversement proportionnelle à sa probabilité d'apparition. Plus cette probabilité est élevée, plus la taille du code est petite. Ainsi, les codes n'ont pas forcément la même taille. Après la génération des codes, les messages sont transcodés. A cause de la taille variable des sous-vecteurs qui les composent, ceux-ci doivent être uniformisés par des bits additionnels générés aléatoirement afin que ces messages puissent être appris.

L'ajout de bits additionnels au code de la valeur d'un symbole conduit à la production à plusieurs codes ayant le même préfixe et étant associés à cette valeur.

Ces codes uniformisés possibles sont similaires aux clones dans un S1SM_I. En effet, ils sont associés à une unique valeur de symbole avec chaque code uniformisé possible s'identifiant à un unique clone.

Au lieu de générer un dictionnaire de codes comme réalisé dans [63], il est possible d'attribuer un certain nombre de clones à un neurone en fonction de la probabilité d'apparition de la valeur de son symbole associé et en fonction du nombre total de clones disponibles par cluster d'un sous-réseau (Ncl).

Soit $n_{(i,j)}$, un neurone et $F_j(a_i)$ la probabilité d'apparition de sa valeur de symbole associée. Soit un réseau S1SM(1,C,L,Ncl). Alors, le nombre de clones $Ncl\left(n_{(i,j)}\right)$ alloués au neurone $n_{(i,j)}$ est:

$$Ncl\left(n_{(i,j)}\right) = \left\lfloor Ncl * P_j(a_i) \right\rfloor + 1 \quad (32)$$

Où $\lfloor \quad \rfloor$ désigne l'arrondi inférieur.

L'avantage d'une telle allocation est de pouvoir déterminer le nombre optimal de clones pour chaque neurone permettant de réduire la variance entre les densités locales à partir du nombre de clones disponibles dans un cluster. L'inconvénient de cette méthode est la nécessiter d'un apriori sur la distribution (tout comme l'approche *huffman* dans [63]). L'objectif des réseaux à allocation dynamique de clones est alors d'*estimer* la distribution durant l'apprentissage car l'allocation est réalisée progressivement en fonction de la densité locale.

d) Apprentissage

L'apprentissage d'un message dans un S1SM nécessite de sélectionner un unique clone actif dans chaque cluster. Dans le cas d'un apprentissage par compétition, il est nécessaire de prendre en compte la densité de chaque clone actif après apprentissage du message courant (Voir Section III.E). Or, dans un S1SM, chaque cluster contient plusieurs clones actifs. Afin de réaliser cette compétition, il faudrait calculer le score pour toutes les combinaisons d'apprentissage possibles (c.à.d. toutes les possibilités d'activation de connexion par le message à apprendre). Pour un S1SM(K,C,L,Ncl), le nombre de combinaisons à évaluer est de $\left(\frac{Ncl}{L}\right)^C$, ce qui n'est pas envisageable. Nous avons donc choisi de faire une sélection aléatoire (ALT) qui offre de bonnes performances comme nous le verrons dans la section dédiée aux expériences.

3. SNS1

a) Définition

Un réseau SNS1 contient un nombre constant de sous-réseaux $\left(\forall i,j,\left|\Gamma\left(n_{(i,j)}\right)\right| = K, K \geq 1\right)$ et un unique clone est alloué à chaque partition de chaque neurone $\left(\forall i,j,\forall p \in \Gamma\left(n_{(i,j)}\right), |p| = 1\right)$.

La Figure III.5 montre l'aperçu d'un SNS1(4,3,3,3) ayant 4 sous-réseaux contenant chacun 3 clusters, avec chaque cluster étant muni de 3 clones, un unique clone étant alloué à chaque neurone. Un SNS1 est alors similaire au regroupement de plusieurs GBNNs au sein d'un même réseau.

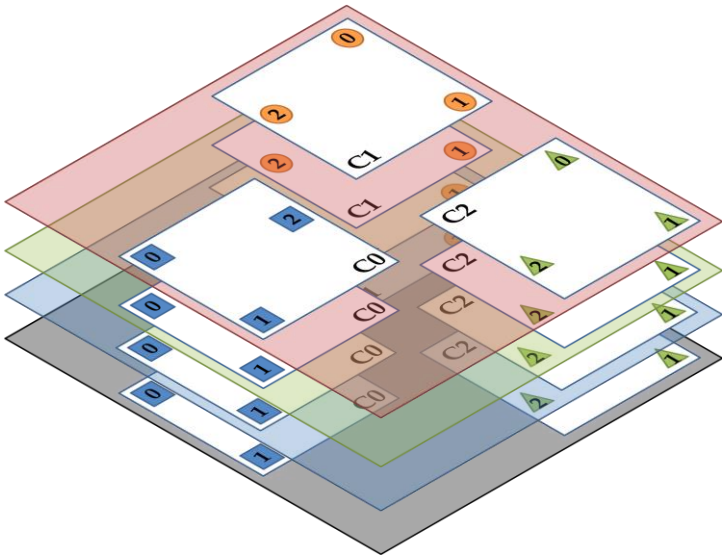


Figure III.5 Schéma d'un SNS1(4,3,3,3)

Pour un SNS1(K,C,L,Ncl), $K \geq 1$ (plusieurs sous-réseaux) et $Ncl = L$ (un unique clone par partition). Son nombre total de poids synaptiques est donné par:

$$|\mathcal{W}| = \frac{K * L^2 * (C-1) * C}{2} \tag{33}$$

Un réseau de neurones contenant plusieurs sous-réseaux est plus communément appelé *ensemble de réseau de neurones (Neural Network Ensemble)* [80]. Dans [80], le réseau est composé de plusieurs réseaux indépendants (les sous-réseaux), chaque sous-réseau étant spécialisé dans la reconnaissance d'un type d'information (classification). Il a été montré qu'un tel type de réseau améliore les performances dans des tâches de classification [81], de généralisation [80] ou encore de prédiction [82]. Dans notre étude, l'objectif est d'augmenter la capacité de stockage en multipliant le nombre de sous-réseaux dans lesquels les messages peuvent être appris.

Dans un réseau SNS1(K,C,L,Ncl), le nombre total de clones pour chaque neurone est égal au nombre de sous-réseaux K. Cette multiplication du nombre de sous-réseaux permet de multiplier le nombre maximal de messages appris par le réseau de base (un GBNN) par le nombre de sous-réseaux.

b) Apprentissage

L'apprentissage dans un SNS1 peut être soit aléatoire (ALT), soit par compétition entre sous-réseaux.

Concernant la compétition, un algorithme *Least Dense Selection* est proposé. Cet algorithme consiste à calculer le score de chaque sous-réseau et de choisir aléatoirement l'un des sous-réseaux ayant le meilleur score c.à.d. le score le plus faible. Le score d'un sous-réseau est la somme des scores des clones actifs au sein de ce sous-réseau.

Lorsqu'un SNS1 reçoit un message à apprendre, les neurones (et leurs clones) associés aux valeurs de symbole appartenant au message deviennent actifs. Soit $n^{x}_{(*,j,k)}$, un clone activé par un message à apprendre. Le score $sc\left(n^{x}_{(*,j,k)}\right)$ de ce clone est donné par :

$$sc\left(n^{x}_{(*,j,k)}\right) = \sum_{j'=0,j'\neq j}^{C-1}\left(\sum_{n^{x'}_{(i'_{,},j',k)}\in\mathcal{N}(j,k)}\left(v\left(n^{x'}_{(i',j',k)}\right)\oplus w\left(n^{x}_{(*,j,k)};\, n^{x'}_{(i',j',k)}\right)\right)\right) \tag{34}$$

Ce score représente le nombre de poids à 1 que possèdera le clone si jamais il apprend le message tout en excluant les poids que l'apprentissage de ce message mettra à 1 alors qu'ils sont déjà à 1 $\left(w\left(n^{x}_{(*,j,k)};\, n^{x'}_{(*,j',k)}\right)\right)$. Il quantifie l'augmentation de la densité locale de ce clone si jamais il apprend le message (nombre de poids qui passeront de 0 à 1) tout en incluant la densité actuelle (nombre de poids à 1 non concernés par le message).

Le score $sc(k)$ du sous-réseau k est la somme des scores de ses clones actifs :

$$sc(k) = \sum_{j=0}^{C-1}\left(sc\left(n^{x}_{(*,j,k)}\right)\right) \tag{35}$$

Ainsi le sous-réseau k_l qui apprend le message est l'un des sous-réseaux possédant un score $sc(k_l)$ égal au score minimal :

$$sc(k_l) = \min\left(\left(sc(k)\right)_{0\leq k<K}\right) \tag{36}$$

Cette compétition permet de choisir le sous-réseau qui aura après l'apprentissage, la plus faible densité locale pour les clones activés tout en prenant en compte que ces clones ont la possibilité de réutiliser les poids déjà mis à 1 (Propriété prise en compte par le XOR ⊕). Ainsi, elle minimise à la fois la densité locale après apprentissage des clones sélectionnés et donc globalement la croissance de la densité globale.

4. SNSM

Un réseau SNSM contient un nombre constant de sous-réseaux $(\forall i,j, |\Gamma(n_{(i,j)})| = K, K \geq 1)$ et un nombre constant de clones est alloué dans chaque partition d'un même neurone $(\forall i,j, \forall p \in \Gamma(n_{(i,j)}), |p| = q)$.

La Figure III.6 montre l'aperçu d'un SNSM(4,3,3,9) ayant 4 sous-réseaux contenant 3 clusters chacun, avec chaque cluster étant muni de 9 clones, 3 clones étant alloué à chaque neurone (Soit 3*3=9 clones).

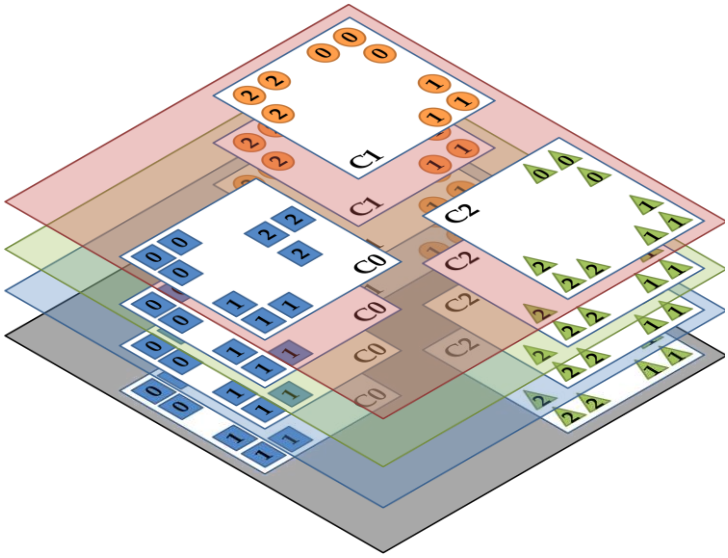


Figure III.6 Aperçu d'un SNSM(4,3,3,9)

Le nombre total de poids synaptiques d'un SNSM(K,C,L,Ncl) est donné par (30).

La variante SNSM est une généralisation des variantes précédentes (S1S1, S1SM et SNS1). Elle possède donc les mêmes déclinaisons en termes de répartition (égale SNSM_E et inégale SNSM_I).

Dans un SNSM_E, un nombre identique de clones est alloué à chaque partition quel que soit le neurone. Dans un SNSM_I, le nombre de clones alloués aux partitions des neurones dépend de la distribution.

En termes d'apprentissage, comme les réseaux S1SM, la compétition est difficile à mettre en place. Ainsi, l'apprentissage se fait par sélection aléatoire (ALT). Pour apprendre un

message au sein d'un SNSM et après l'activation des clones associés aux symboles connus du message à apprendre, un sous-réseau est sélectionné de manière aléatoire et un unique clone actif est sélectionné aléatoirement dans chaque cluster de ce sous-réseau.

La structure du SNSM (plusieurs clones par partition et plusieurs sous-réseaux) lui confère les mêmes propriétés que les variantes S1SM et SNS1. En effet, tandis que la présence de plusieurs clones d'un neurone dans le même sous-réseau tend à uniformiser les densités locales des clones (comme dans S1SM_I), la multiplication du nombre de sous-réseaux tend à multiplier le nombre de messages que le réseau peut apprendre (comme dans SNS1).

Le coût de mémorisation d'un SNSM est donné par l'équation (30).

5. Diversité des réseaux à allocation statique de clones

Dans cette section, les différentes variantes de réseau à allocation statique de clones sont comparées en analysant leur *diversité* qui est le nombre maximal de messages pouvant être appris s'ils ont une distribution uniforme.

Sachant que $Nt = L * C$ est égal au nombre de neurones dans le sous-réseau d'un S1S1, la diversité d'un tel réseau est égale à celle d'un GBNN [62] :

$$Div(S1S1) = \frac{(C-1) * Nt^2}{2 * C^2 * \log_2\left(\frac{Nt}{C}\right)} = \frac{(C-1) * (L)^2}{2 * \log_2(L)} \tag{37}$$

Il est possible de définir la diversité pour les autres variantes de réseau à allocation statique de clones.

Pour un $S1SM(1, C, L, Ncl)$, le nombre total de clones par sous-réseau est $Nt = Ncl * C$. La diversité est alors donnée par :

$$Div(S1SM) = \frac{(C-1) * (Ncl)^2}{2 * \log_2(Ncl)} \tag{38}$$

Pour un $SNS1(K, C, L, L)$, le nombre de sous-réseaux multiplie linéairement la diversité d'un S1S1 par K. Celle-ci est égale à :

$$Div(SNS1) = K * \frac{(C-1) * (L)^2}{2 * \log_2(L)} \tag{39}$$

Pour un $SNSM(K', C, L, Ncl')$, le nombre de sous-réseaux multiplie linéairement la diversité par K' et le nombre de clones par cluster est multiplié par $\frac{Ncl'}{L}$ par rapport à un S1S1. Sa diversité est donnée par :

$$Div(SNSM) = K' * \frac{(C-1) * (Ncl')^2}{2 * \log_2(Ncl')} \quad (40)$$

Afin que ces différents réseaux aient le même coût de mémorisation, il faut que la condition suivante soit respectée :

$$K = \left(\frac{Ncl}{L}\right)^2 = K' * \left(\frac{Ncl'}{L}\right)^2 \quad (41)$$

Cette condition est obtenue en égalisant leur différent coût de mémorisation (30), (31) et (33).

La comparaison des diversités des différentes variantes (Voir (42) et (43)) montre que la diversité d'un SNS1 est toujours supérieure à celle d'un S1SM (car $Ncl > L$). Cette différence augmente en fonction du nombre de clones par cluster, Ncl. Le SNS1 surpasse aussi le SNSM mais avec un différence plus réduite (car $Ncl > Ncl'$).

$$\frac{Div(SNS1)}{Div(S1SM)} = \frac{K * L^2 * \log_2(Ncl)}{(Ncl)^2 * \log_2(L)} = \frac{\log_2(Ncl)}{\log_2(L)} > 1 \quad (42)$$

$$\frac{Div(SNS1)}{Div(SNSM)} = \frac{K * \log_2(Ncl')}{K' * \left(\frac{Ncl'}{L}\right)^2 * \log_2(L)} = \frac{\log_2(Ncl')}{\log_2(L)} > 1 \quad (43)$$

La variante SNS1 est donc celle qui est capable d'apprendre le plus de messages. La différence de diversité entre un SNS1 et un S1SM (et entre un SNS1 et un SNSM) augmente en fonction de Ncl (resp. Ncl'), le nombre total de clones par cluster. Cependant, la diversité d'un réseau ne garantit par la performance. En effet, la distribution des messages à apprendre joue aussi un rôle. Dans le cas d'une distribution garantissant une performance optimale (c.à.d. la distribution uniforme), la variante SNS1 est celle qui obtiendra la meilleure performance.

E. Réseaux à allocation dynamique de clones

Dans un réseau à allocation dynamique de clones, les clones peuvent être alloués aux neurones et les sous-réseaux peuvent être ajoutés au réseau au fur et à mesure de l'apprentissage et en fonction du besoin.

Pour rester dans un cadre réaliste et en vue des comparaisons avec l'état de l'art, la taille mémoire utilisable par le réseau est limitée. Cela signifie que l'*allocation infinie* est rendue

impossible. Ainsi, le nombre de maximal de clones par cluster et le nombre maximal de sous-réseaux sont limités.

A cause de ces contraintes, certaines variantes de réseau à clones deviennent identiques et ne sont pas simulées pour maintenir la clarté des comparaisons.

Dans la variante DNSM, le nombre de sous-réseaux peut augmenter (**DN**SM) tandis que chaque partition contient un nombre constant de clones (DN**SM**). Si le nombre de sous-réseaux est limité, alors cette variante est similaire à un SNSM qui contient un nombre de sous-réseaux constant. Il a été étudié dans le CHAPITRE III.D.4.

Dans la variante DNDM, le nombre de sous-réseaux peut augmenter (**DN**DM) tandis que chaque partition contient un nombre de clones pouvant augmenter (DN**DM**). Si le nombre de sous-réseaux est limité, alors cette variante est similaire à un SNDM qui contient un nombre constant de sous-réseaux.

Dans la variante DNS1, le nombre de sous-réseaux peut augmenter (**DN**S1) tandis que chaque partition contient un unique clone (DN**S1**). Si le nombre de sous-réseaux est limité, alors cette variante est similaire à un SNS1 qui contient un nombre constant de sous-réseaux. Il a été étudié dans le CHAPITRE III.D.3.

Ainsi, dans cette section, seules deux variantes sont étudiées : le S1DM et le DNDM.

1. S1DM

a) Définition

Un réseau S1DM contient un unique sous-réseau $\left(\forall i,j,\left|\Gamma\left(n_{(i,j)}\right)\right|=1\right)$ et le nombre de clones alloués aux partitions peut augmenter au cours de l'apprentissage $\left(\forall i,j,\forall p\in\Gamma\left(n_{(i,j)}\right),|p|\leq Ncl\right)$. La variante S1DM est l'équivalent dynamique de S1SM (Voir CHAPITRE III.D.2).

La Figure III.7 montre l'aperçu d'un S1DM(1,3,3,9) ayant un unique sous-réseau contenant 3 clusters, avec chaque cluster étant muni de 9 clones, 3 clones en moyenne pouvant être alloués chaque neurone.

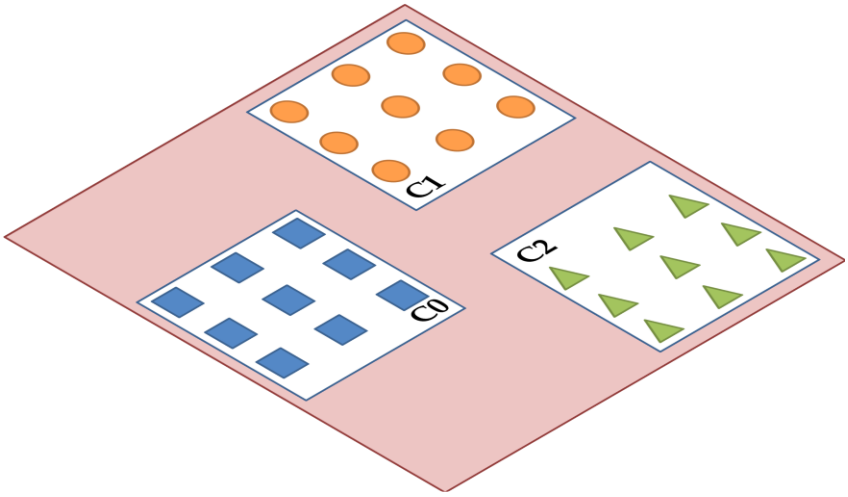


Figure III.7 Schéma d'un S1DM(1,3,3,9)

b) Apprentissage

La Figure III.8 montre l'algorithme d'apprentissage d'un message dans un S1DM.

Dans cet algorithme, l'allocation d'un clone à un neurone au sein d'un cluster est possible si et seulement s'il reste des clones non allouées dans ce cluster.

Si cette condition est vérifiée, l'allocation d'un clone à un neurone actif est réalisée lorsque : (1) soit la densité locale du dernier clone alloué à ce neurone est supérieure à un seuil (valeur arbitraire), (2) soit s'il n'existe aucun clone de ce cluster alloué à ce neurone.

Le déroulement de l'algorithme est le suivant : lorsque le réseau reçoit un message, les clones associés aux valeurs de symboles dans le message sont activés. Pour chaque cluster de l'unique sous-réseau, un clone doit être sélectionné. Deux cas de figure sont possibles :

- Il existe aucun clone actif dans le cluster : cela signifie qu'aucun clone n'a été alloué au neurone actif dans ce cluster. Si l'allocation est possible, celle-ci est réalisée et le clone alloué est sélectionné pour l'apprentissage. Dans le cas contraire, l'apprentissage ne peut être réalisé.
- Il existe au moins un clone actif dans le cluster. Ainsi, si la densité du dernier clone utilisé est supérieure à un seuil et si l'allocation est possible, un nouveau clone est alloué au neurone et ce clone est sélectionné. Si la densité est inférieure alors c'est ce dernier clone utilisé qui est sélectionné. Si l'allocation doit être réalisée mais est impossible à faire, alors le clone à sélectionner pour l'apprentissage est choisi aléatoirement.

Ainsi dans les réseaux à allocation dynamique, l'apprentissage peut être un échec si l'allocation des clones est limitée. Comme, il sera montré dans la section dédiée aux expériences, cet inconvénient peut se révéler utile car l'échec d'un apprentissage signifie que

la densité n'augmente pas ce qui peut être utile pour garantir une bonne performance. Il est possible de rendre un apprentissage toujours possible en allouant au moins un clone dans chaque partition de chaque neurone. Il s'agit d'une *pré-allocation*.

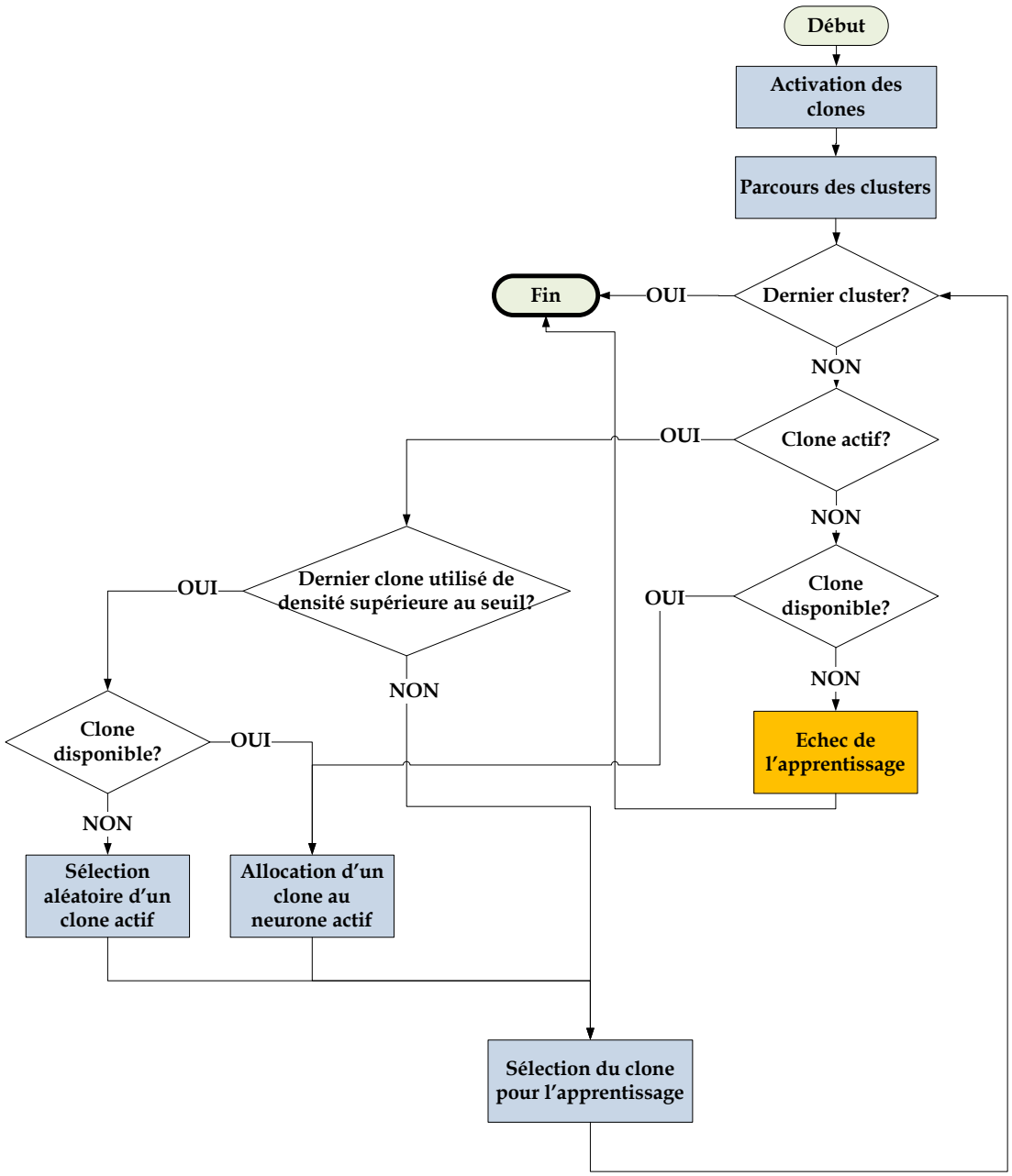


Figure III.8 Algorithme d'apprentissage du S1DM

2. DNDM

a) Définition

Dans un réseau DNDM, le nombre de sous-réseaux peut augmenter $\left(\forall i,j,\left|\Gamma\left(n_{(i,j)}\right)\right| \leq K, K \geq 1\right)$ et des clones non alloués dans chaque cluster peuvent être alloués dans les

partitions lors de l'apprentissage d'un message $(\forall i,j,\forall p \in \Gamma(n_{(i,j)}), |p| \leq Ncl)$. La variante DNDM est l'équivalent dynamique de SNSM (Voir CHAPITRE III.D.4).

La Figure III.9 montre l'aperçu d'un DNDM(4,3,3,9) ayant 4 sous-réseaux contenant chacun 3 clusters, avec chaque cluster étant muni de 9 clones, 3 clones en moyenne pouvant être alloués chaque neurone.

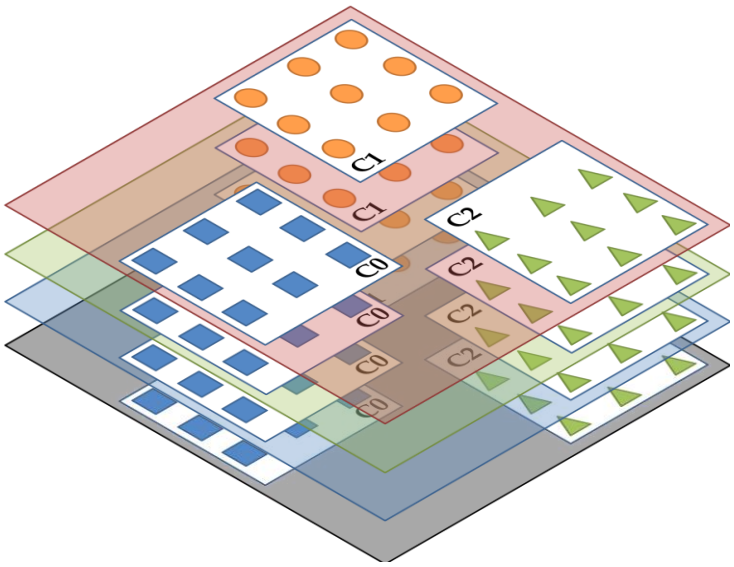


Figure III.9 Aperçu d'un DNDM(4,3,3,9)

b) Apprentissage

La Figure III.10 montre l'algorithme d'apprentissage d'un message dans un DNDM.

Dans cet algorithme, l'allocation d'un clone à un neurone au sein d'un cluster est possible si et seulement s'il reste des clones non alloués dans ce cluster.

Si cette condition est vérifiée, l'allocation d'un clone à un neurone actif est réalisée lorsque : (1) soit la densité locale du dernier clone alloué à ce neurone est supérieure à un seuil (valeur arbitraire), (2) soit s'il n'existe aucun clone de ce cluster alloué à ce neurone.

L'ajout d'un sous-réseau au réseau est réalisé lorsque le nombre maximal de sous-réseaux utilisable n'a pas été atteint et si l'allocation d'un clone à un neurone actif n'a pas pu être réalisée dans le dernier sous-réseau utilisé.

Lors de l'apprentissage des messages, les sous-réseaux sont remplis progressivement jusqu'à atteindre la limite en termes de nombre maximal de sous-réseaux. Quand ce nombre n'est pas atteint, le sous-réseau en cours essaie d'apprendre le message en utilisant le même algorithme que dans S1SM.

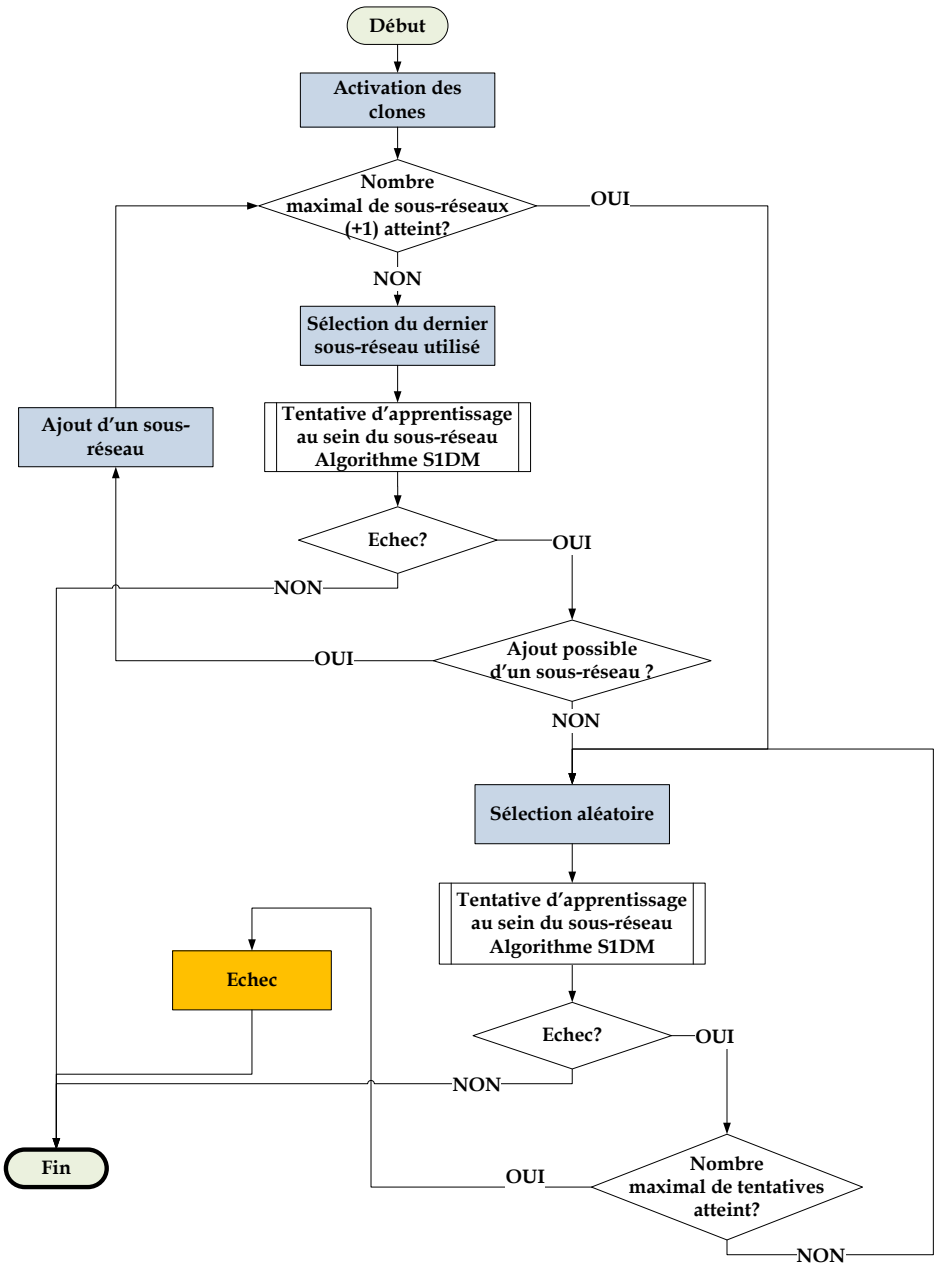


Figure III.10 Algorithme d'apprentissage d'un message dans un DNDM

Un sous-réseau est ajouté lorsque l'allocation d'un clone est impossible et si le nombre maximal de sous-réseaux n'est pas atteint. Dans le cas contraire, un sous-réseau est sélectionné aléatoirement. Ensuite, un clone actif est choisi dans chaque cluster de manière aléatoire. Dans ce cas, l'apprentissage est un échec s'il existe au moins un cluster ne contenant pas de clone actif. Plusieurs tentatives peuvent être réalisées (sélection aléatoire d'un autre sous-réseau et tentative d'apprentissage). Dans les expériences, une seule tentative a été réalisée. Il est aussi possible de faire une recherche exhaustive d'un sous-réseau capable d'apprendre le message mais ce dernier cas n'a pas été étudié.

3. Diversité des réseaux à allocation dynamique de clones

La diversité des réseaux à allocation dynamique de clones est identique à leur équivalent statique. En effet, chaque clone du réseau permet l'apprentissage de la même quantité d'informations (indépendamment de l'allocation). De plus, étant donné que la diversité est définie uniquement dans le cadre de messages ayant une distribution uniforme, tous les neurones auront reçus le même nombre de clones à la fin de l'apprentissage.

F. Expériences

Les réseaux étudiés ont été évalués par des simulations consistant au décodage de messages ayant des distributions non-uniformes. Pour tous les réseaux évalués, le nombre de clusters par sous-réseau est de 8 et le nombre de neurones par cluster est de 32. La distribution des messages à apprendre est gaussienne avec une moyenne de 16 et un écart-type de 5.

Tous les réseaux ont été configurés pour avoir le même coût de mémorisation égal à 16 GBNN(8,32). Le ratio 16 est identique à celui utilisé par l'état de l'art pour comparer leurs différentes approches [63]. Le coût mémoire des matrices d'adjacence des réseaux est alors égal à 917kbits. L'évaluation des réseaux se base sur le décodage de messages ayant été aléatoirement sélectionnés et aléatoirement effacés à 50%. Cela implique que dans chaque message décodé, 4 symboles sur 8 sont inconnus et qu'au début du décodage d'un message par un réseau, 4 clusters sur 8 sont inconnus dans chaque sous-réseau. 4 itérations ont été réalisées lors de chaque décodage car c'est le nombre permettant d'obtenir la performance maximale d'un GBNN [62].

Dans toutes les figures, les performances des GBNNs/SIS1s (en distribution uniforme et en distribution gaussienne) sont données comme références.

La détermination des meilleurs modèles se déroule en trois comparaisons :

- Comparaison des performances entre réseaux à allocation statique de clones ;
- Comparaison des performances entre réseaux à allocation dynamique de clones ;
- Comparaison des performances entre les meilleurs réseaux à allocation statique de clones et les meilleurs réseaux à allocation dynamique de clones.

Dans les figures montrées, et en cas d'ambiguïté sur le meilleur réseau, les réseaux évalués seront comparés selon trois domaines de performance :

- Hautes performances ($TED \leq 5\%$) ;
- Moyennes performances ($5\% < TED \leq 30\%$) ;
- Basses performances ($TED > 30\%$).

1. Performances des réseaux à allocation statique de clones

Le Tableau 2 montre les paramètres des différentes variantes de réseaux à allocation statique de clones évaluées.

Tableau 2 Paramètres des réseaux à allocation statique de clones évalués

Variantes	Nombre de sous-réseaux : K	Nombre total de clones dans un cluster d'un sous-réseau : Ncl	Nombre moyen de clones par neurone dans un sous-réseau :
S1SM	1	128	128/32=4
SNS1	16	32	32/32=1
SNSM	4	64	64/32=2

De nombreuses variantes de réseaux à allocation statique de clones ont été présentées. Chacune est munie de configurations différentes en fonction de l'algorithme d'apprentissage (SNS1) ou encore du mode de répartition (S1SM et SNSM). Afin de ne pas surcharger les figures avec de nombreuses courbes, des comparaisons sont réalisées progressivement en mettant face à face les configurations pour la même variante. Ainsi, la comparaison des meilleurs réseaux à allocation statique de clones est réalisée à la fin.

a) Performances des réseaux S1SM

La Figure III.11 montre la performance des réseaux S1SM (état de l'art [63]) selon les différents modes de répartitions des clones (égale, inégale). La meilleure performance est obtenue par le réseau S1SM_I (*huffmann*) dont la répartition des clones s'adapte à la distribution et réduit la variance entre les densités locales des clones.

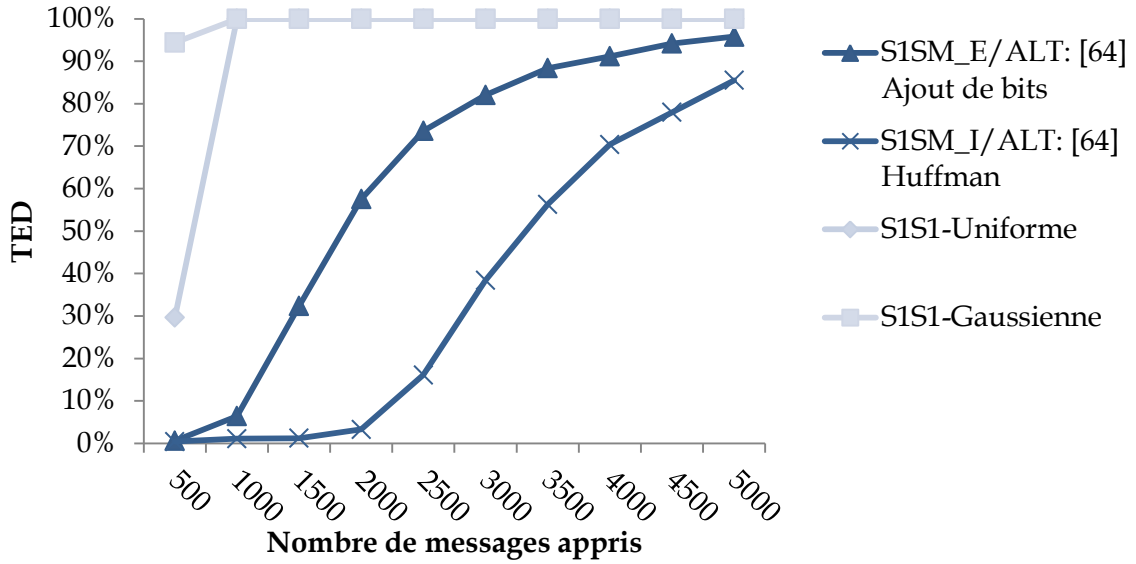


Figure III.11 Performances des réseaux S1SM selon les modes de répartition des clones

b) *Performances des réseaux SNS1*

La Figure III.12 montre les performances des réseaux SNS1 selon les différents algorithmes d'apprentissage présentés : ALéaToire (ALT) et Least Dense Selection (LDS). Le réseau le plus performant est le SNS1/LDS. Ainsi, l'algorithme compétitif proposé améliore la performance. Cette amélioration est due à la tendance de cet algorithme à minimiser l'augmentation de la densité locale des clones du réseau.

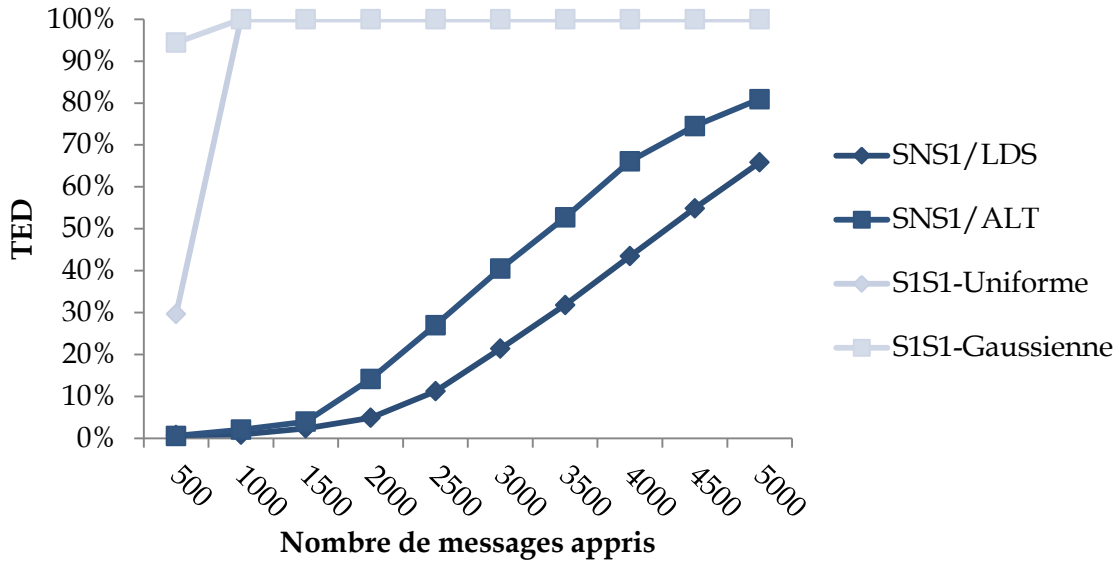


Figure III.12 Performances des réseaux SNS1 selon les différents algorithmes d'apprentissage

c) *Performances des réseaux SNSM*

La Figure III.13 montre la performance des réseaux SNSM selon les différents modes de répartition des clones. La meilleure performance est obtenue par le réseau SNSM_I dont la répartition des clones s'adapte à la distribution.

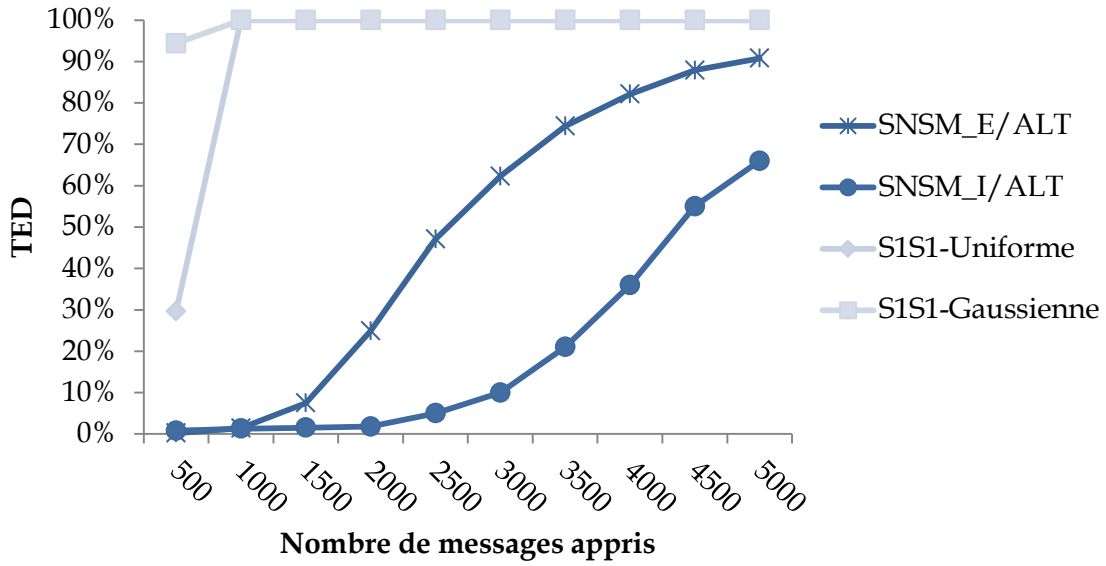


Figure III.13 Performances des réseaux SNSM selon les modes de répartition des clones

d) Comparaison des meilleurs réseaux à allocation statique de clones

La Figure III.14 montre la performance des meilleures variantes de réseaux à allocation statique de clones.

Dans un domaine de hautes performances, la variante SNSM_I/ALT est capable d'apprendre le plus de messages (2500 messages). Elle est concurrencée par le S1SM_I/ALT (2000 messages).

Dans un domaine de moyennes performances, la variante SNSM_I/ALT est capable d'apprendre le plus de messages (3500 messages). Elle est concurrencée par le SNS1/LDS (3000 messages).

Dans un domaine de basses performances, la variante SNSM_I/ALT est capable d'apprendre le plus de messages. Elle est concurrencée par le SNS1/LDS.

En somme, le réseau le plus performant est le SNSM_I. Les réseaux SNS1 et SNSM obtiennent de meilleures performances par rapport aux réseaux de l'état de l'art (S1SM_I et S1SM_E) [63]. Ces résultats montrent que l'utilisation de plusieurs sous-réseaux et/ou d'un algorithme d'apprentissage compétitif permettent d'obtenir de bonnes performances. Tandis que la présence de plusieurs sous-réseaux multiplie le nombre de messages pouvant être appris, l'algorithme compétitif ralentit l'augmentation de la densité globale ce qui freine la baisse de performance. Cependant, l'allocation des clones aux neurones en fonction de la distribution (S1SM_I et SNSM_I) joue aussi un rôle dans les performances car elle réduit la variance entre les densités locales.

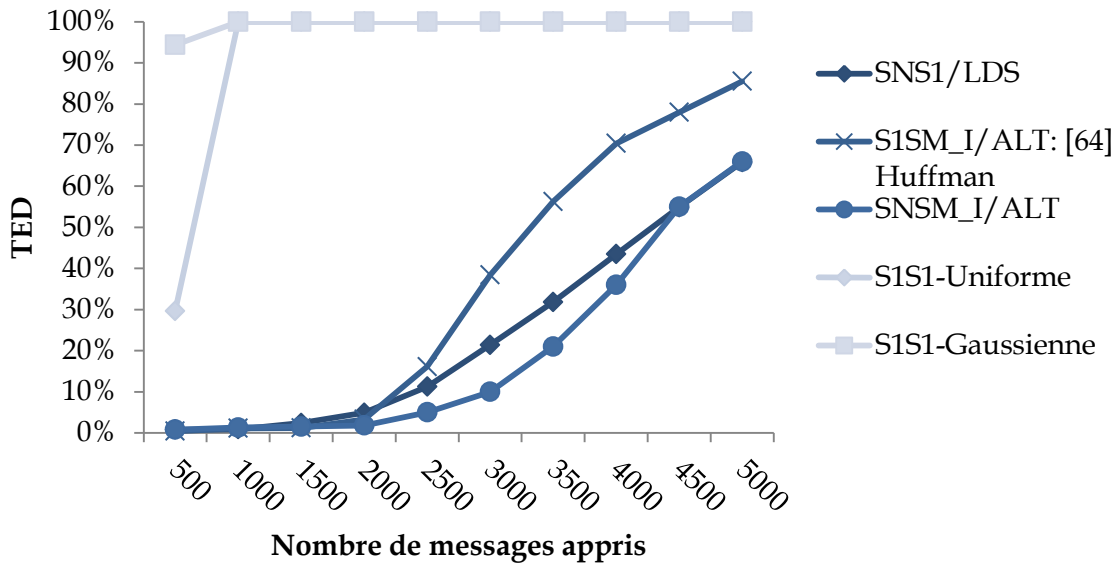


Figure III.14 Performances des meilleurs réseaux à allocation statique de clones

Les réseaux statiques sont performants mais uniquement s'ils sont configurés avec un apriori sur la distribution (SNSM_I). Cependant, si la distribution est inconnue, le réseau ne

peut être configuré de manière optimale. Ainsi, l'allocation dynamique de clones a pour objectif principal de permettre au réseau de s'adapter à la distribution des messages afin d'obtenir les mêmes performances que les meilleurs réseaux nécessitant un apriori sur la distribution (SNSM_I).

2. Performances des réseaux à allocation dynamique de clones

Dans le cadre des expériences concernant les réseaux à allocation dynamique de clones, le nombre maximal de clones par cluster et le nombre maximal de sous-réseaux sont limités.

Le Tableau 3 montre les configurations des différentes variantes de réseaux à allocation dynamique de clones évaluées.

Tableau 3 Configuration des variantes de réseaux à clones dynamique évalués

Variantes	**Nombre de sous-réseaux : K**	**Nombre maximal de clones dans un cluster d'un sous-réseau : Ncl**	**Nombre moyen de clones par neurone dans un sous-réseau :**
S1DM	1	128	128/32=4
DNDM	4	64	64/32=2

La pré-allocation (PREALLOC) réserve un clone par neurone pour chaque sous-réseau (un clone par partition, 32 clones par cluster d'un sous-réseau).

De manière similaire à l'évaluation des réseaux à allocation statique de clones, les comparaisons sont réalisées selon chaque variante et pour différentes configurations. Ainsi, la comparaison des meilleurs réseaux est réalisée à la fin. Les configurations ici sont la valeur du seuil et l'usage de la pré-allocation dans le réseau. Dans les figures, la valeur $\alpha = H\%$ dans la légende représente le seuil. Celui-ci a été choisi comme étant inférieur à la densité maximale du GBNN permettant d'obtenir de bonnes performances ($\alpha < 30\%$).

a) Performances des réseaux S1DM

La Figure III.15 montre les performances des réseaux S1DM pour différentes configurations en termes de seuil et d'usage de la pré-allocation au sein du réseau. La meilleure performance est obtenue par le S1DM(5%).

La pré-allocation est inutile dans les S1DMs car ceux-ci disposent de suffisamment de clones pour permettre l'allocation d'au moins un clone à chaque neurone. Tous les S1DMs utilisant la pré-allocation ont presque la même tendance que leur équivalent sans pré-allocation, raison pour laquelle une seule courbe a été insérée. On constate aussi qu'un seuil élevé (S1DM, 20%) conduit automatiquement à une performance très faible (proche d'un TED de

100%.) car les clones sont alloués trop lentement. Ainsi, ils sont trop surchargés et leur densité est élevée. D'un autre côté, les S1DMs ayant des seuils faibles ont une performance similaire au S1DM(5%). Ce résultat est dû à leur nombre important de clones qui implique une certaine similarité dans leur allocation.

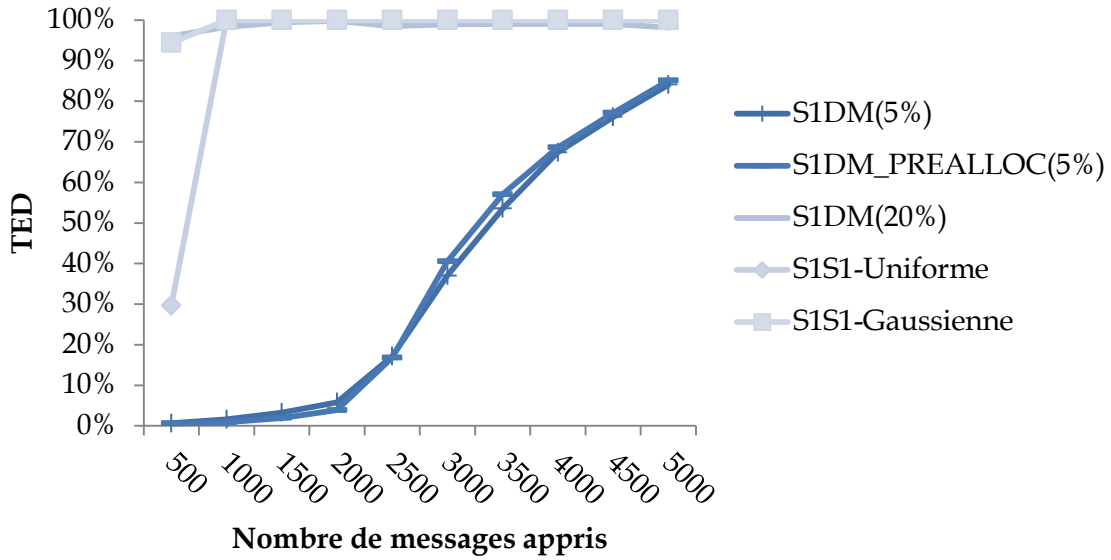


Figure III.15 Performances des réseaux S1DM selon le seuil et l'usage de la pré-allocation

b) Performances des réseaux DNDM

La Figure III.16 montre les performances des DNDMs pour différentes configurations en termes de seuil et de l'usage de la pré-allocation au sein du réseau. La meilleure performance est obtenue par le DNDM(10%).

Il existe une différence de performance entre DNDM_PREALLOC(5%) et DNDM(5%). Cette différence est due à la pré-allocation au sein du DNDM_PREALLOC qui réserve 32 clones sur 64. Tandis que DNDM(5%) provoque beaucoup d'échecs d'apprentissage à cause d'un seuil élevé, DNDM_PREALLOC(5%) assure que l'apprentissage soit toujours possible mais il ne peut allouer qu'un faible nombre de clones.

On constate aussi qu'un seuil élevé (DNDM, 20%) conduit à de mauvaises performances et un seuil très faible (DNDM, 5%) conduit à une performance très variable. En effet, tandis qu'un seuil élevé conduit à une lente création de clones et permet de toujours apprendre un message, un seuil faible conduit à une création rapide de clones et cause beaucoup d'échecs d'apprentissage si le nombre de clones disponible est faible. Cependant, l'échec de l'apprentissage d'un message n'augmente pas la densité et peut ralentir la baisse de la performance.

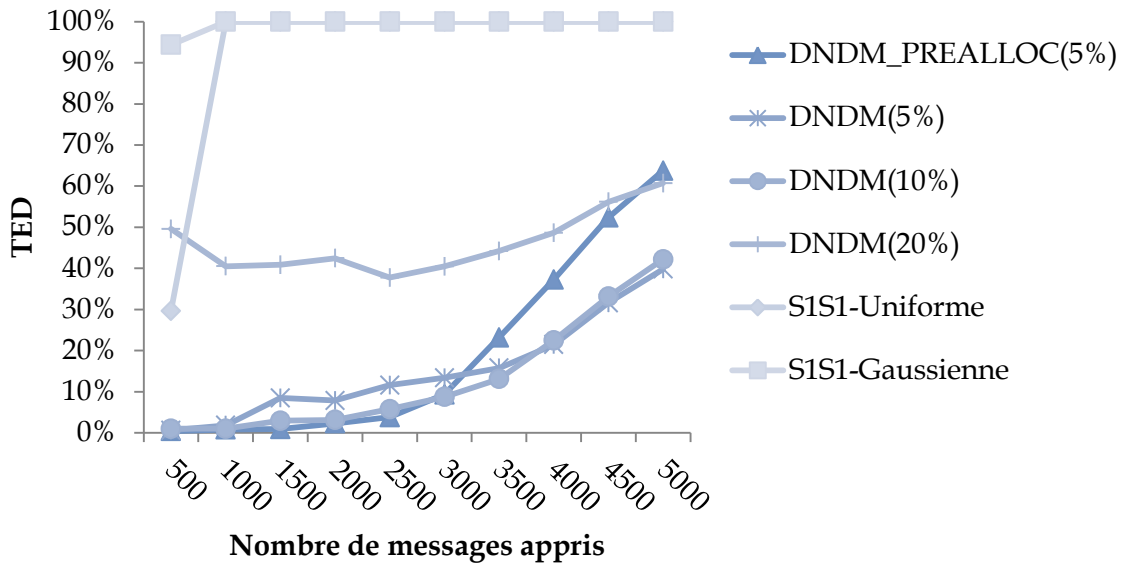


Figure III.16 Performances des réseaux DNDM selon le seuil et l'usage de la pré-allocation

c) Comparaison des meilleurs réseaux à allocation dynamique de clones

La Figure III.17 montre la performance des meilleurs réseaux à allocation dynamique de clones. La variante DNDM(10%) est la variante la plus performante. Les résultats montrent une fois de plus que la présence de plusieurs sous-réseaux permet d'obtenir de meilleures performances.

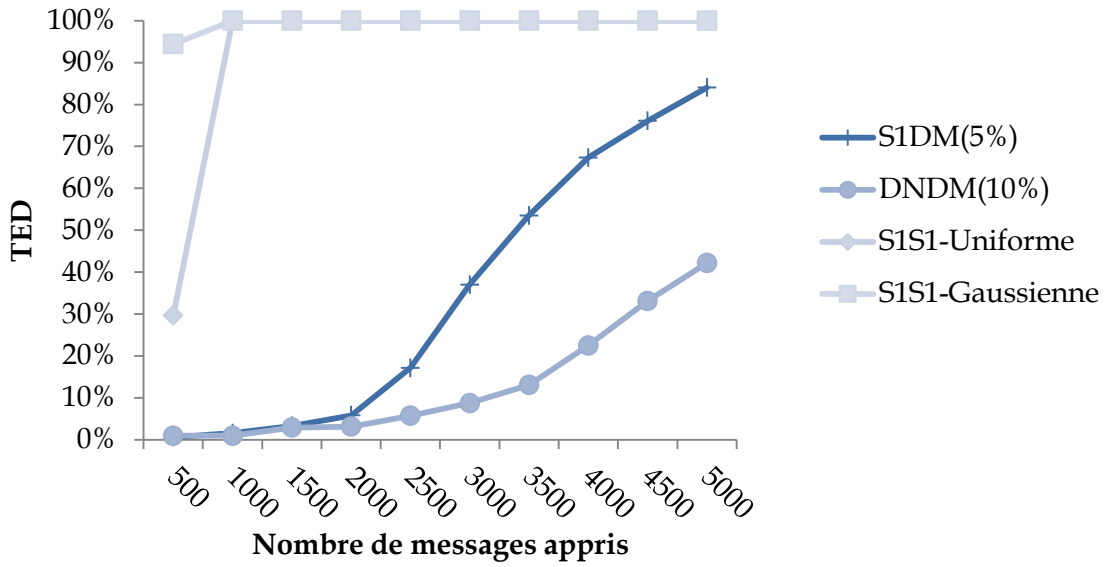


Figure III.17 Performances des meilleurs réseaux à allocation dynamique de clones

3. Réseaux Statiques VS Réseaux Dynamiques

La Figure III.18 montre la performance des meilleures variantes de réseaux à clones.

Dans un domaine de hautes performances, la variante SNSM_I/ALT est capable d'apprendre le plus de messages (2500 messages). Elle est concurrencée par le DNDM(10%) (2000 messages).

Dans un domaine de moyennes performances, la variante DNDM(10%) est capable d'apprendre le plus de messages (4000 messages). Elle est concurrencée par le SNSM_I/ALT (3500 messages).

Dans un domaine de basses performances, la variante DNDM(10%) est capable d'apprendre le plus de messages. Elle est concurrencée par le SNS1/LSD.

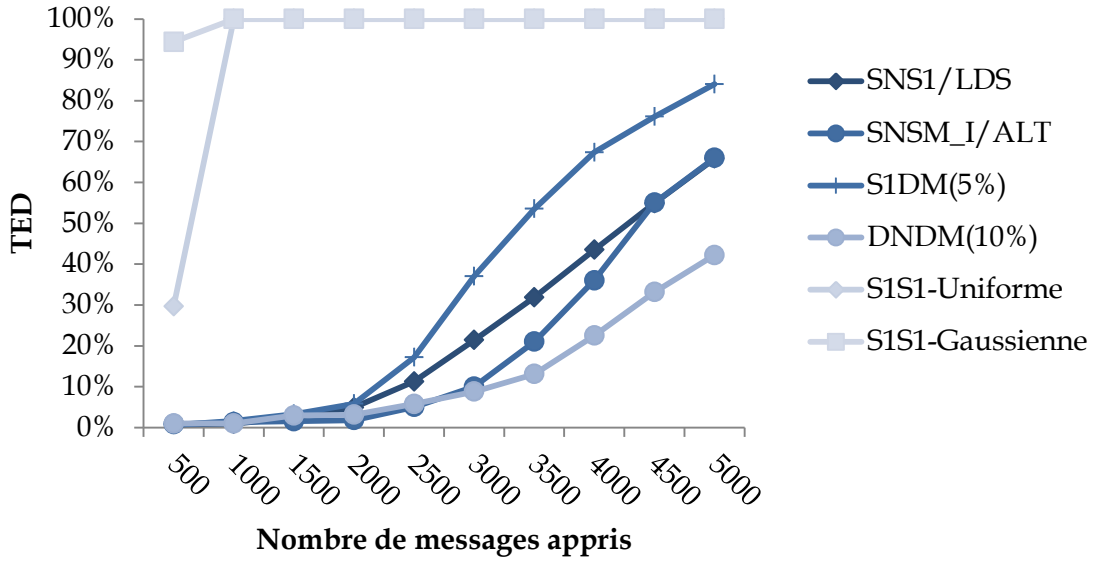


Figure III.18 Performances des meilleurs réseaux à clones

En conclusion, le réseau DNDM(10%) est le réseau ayant la meilleure performance parmi toutes les variantes.

Les résultats des évaluations des réseaux à clones montrent que les meilleures variantes possèdent :

- Soit plusieurs sous-réseaux qui permettent d'avoir une grande capacité de stockage (SNS1, SNSM, DNDM) ;
- Soit un algorithme compétitif qui permet de ralentir l'augmentation de la densité locale et globale (SNS1/LDS) ;
- Soit une allocation dynamique de clones qui permet d'allouer les clones aux neurones en fonction de la distribution et au fur et à mesure de l'apprentissage (DNDM).

Il serait donc intéressant de concevoir un modèle intégrant toutes ces caractéristiques : modèle avec allocation dynamique de clones, contenant plusieurs sous-réseaux et muni d'un algorithme compétitif.

Par rapport à l'état de l'art [63] (S1SM_I et S1SM_E), de nombreuses variantes se montrent supérieures en performance (SNSM_I, DNDM(10%)) alors que toutes les réseaux évalués possèdent le même coût de mémorisation.

G. Bilan

Dans ce chapitre, nous avons présenté des modèles permettant d'améliorer les performances des réseaux GBNN pour des messages ayant une distribution non-uniforme.

Ces modèles ont des structures et des algorithmes d'apprentissage différents mais utilisent le même algorithme de décodage. Les résultats des évaluations montrent que les modèles les plus performants sont ceux qui utilisent une allocation dynamique de clones et qui possèdent plusieurs sous-réseaux. L'allocation dynamique permet de s'adapter à la distribution tandis que l'utilisation de plusieurs sous-réseaux permet de multiplier le nombre de messages qu'un réseau peut apprendre. Les résultats montrent aussi que des algorithmes d'apprentissage intelligents (LDS) peuvent se montrer compétitifs par rapport aux algorithmes simples (ALT). Cependant, ils ne sont pas évidents à mettre en œuvre et des études approfondies sont nécessaires.

La spécification de nouveaux modèles nécessite de proposer des architectures matérielles les mettant en œuvre. Dans les chapitres suivants, l'attention est portée sur les architectures matérielles. Après avoir réalisé un état de l'art sur les architectures matérielles mettant en œuvre les réseaux GBNN, des améliorations sont présentées et ils permettent de : (1) simplifier le modèle GBNN et le décodage de messages partiellement effacés et (2) de réduire le coût des architectures existantes. De plus, une architecture générique permettant de manipuler des réseaux à clones de différentes tailles tout en utilisant les mêmes ressources, est proposée.

CHAPITRE IV. Etat de l'Art des Architectures Matérielles de Réseaux à Clusters Binaires

Sommaire du Chapitre :

Dans ce chapitre, l'état de l'art concernant les architectures matérielles mettant en œuvre les réseaux GBNN est présenté tandis que les calculs et la méthode de stockage des poids synaptiques des réseaux à clones sont analysés. Après avoir assimilé les calculs à des multiplications de matrices parcimonieuses par des vecteurs, une architecture performante réalisant ce type calcul est présentée et sert de base pour concevoir une architecture générique mettant en œuvre les réseaux à clones.

A. Introduction

Les architectures matérielles mettant en œuvre des réseaux de neurones peuvent être réparties en deux classes :

- Les *architectures spécialisées* mettant en œuvre des réseaux prédéfinis à l'avance en termes de composition et d'organisation : nombre de neurones, interconnections entre neurones et organisation exclusivement auto-associatif ou hétéro-associatif.
- Les *architectures génériques* mettant en œuvre des réseaux non prédéfinis à l'avance et donc pouvant manipuler des réseaux de tailles variables avec des organisations différentes.

Les architectures génériques sont plus communément appelées FPNA (Field Programmable Neural Array) [83]–[85]. Elle peuvent être mises en œuvre sur des plateformes ASIC (Application-Specific Integrated Circuit) [84], ou encore FPGA [83]. Selon [83], un FPNA est essentiellement défini par deux composantes :

- L'ensemble de ressources représentant des neurones logiques et devant être associés à des neurones physiques ;
- L'ensemble d'interconnexions entre ces ressources.

Dans ce chapitre, nous présentons l'état de l'art des architectures mettant en œuvre le GBNN. L'attention sera portée sur l'analyse du stockage et des calculs qui interviennent au sein des réseaux GBNN et plus globalement au sein des réseaux à clones. Ces analyses visent à fournir des pistes pour concevoir des architectures génériques.

B. Architectures matérielles mettant en œuvre les réseaux GBNN

Plusieurs architectures ont été proposées pour mettre en œuvre le GBNN sur plateforme reconfigurable FPGA (Field Programmable Gate Array).

1. Architectures parallèles

La première architecture [64] se base sur la proposition originelle [62]. Il s'agit d'une architecture ayant les caractéristiques suivantes :

- Calculs et communications totalement parallèles : c.à.d. que toutes les valeurs d'activations ainsi que les poids synaptiques sont transmis en un cycle, les scores de tous les neurones sont calculés en parallèle et les valeurs d'activation de tous les neurones sont obtenues en un cycle.

- Stockage de la matrice d'adjacence du GBNN dans des registres sans tenir compte de la symétrie des connexions entre les neurones de différents clusters. Ainsi, le coût du stockage de la matrice est le double du nombre total de poids synaptiques dans le modèle GBNN.
- Décodage par SOS + WTA (Sum of Sum + Winner Take All, Voir CHAPITRE II.C.5.a)(3)).

L'architecture est principalement composée par les clusters interconnectés dans un réseau maillé où chaque cluster est directement connecté à tous les autres clusters.

La Figure IV.1 présente une vision simplifiée de l'architecture d'un cluster. Un cluster est principalement composé d'un module d'apprentissage et d'un module de décodage. L'architecture est paramétrée en fonction de taille du réseau (Nombre de clusters C et nombre de neurones par cluster L). Ainsi, le module d'apprentissage stocke les poids entre chaque couple de neurones c.à.d. $L^2 * (C-1)$ valeurs car chaque neurone peut être connecté à $L * (C-1)$ neurones distants et chaque cluster contient L neurones. Le module de décodage réalise l'étape de calcul du score et les étapes du WTA.

[65] propose une architecture améliorée basée sur un décodage avec l'utilisation d'un SOM (Sum Of Max, Voir CHAPITRE II.C.5.a)(3)) à la place de SOS. Cette architecture conserve les mêmes propriétés que l'architecture proposée dans [64] (calculs et communications en parallèle, stockage des poids sans tenir compte de la symétrie des connexions).

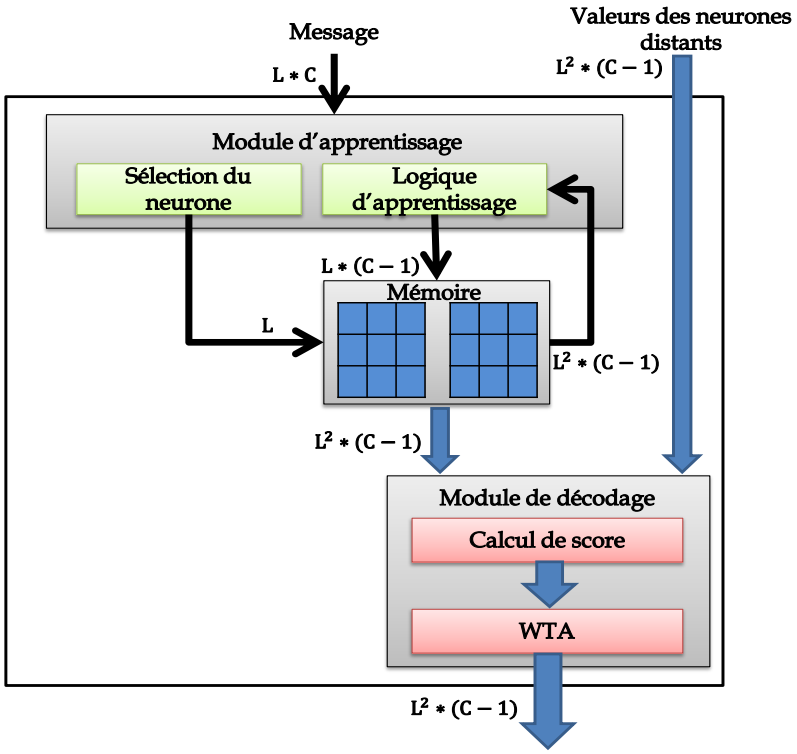


Figure IV.1 Architecture simplifié d'un cluster

2. Architecture sérialisée

Une architecture sous forme de NOC (Network On Chip) a aussi été proposée pour le GBNN [86].

La Figure IV.2 montre un aperçu de cette architecture. Celle-ci réalise un calcul SOS+WTA et stocke la matrice en tenant compte de la symétrie des poids synaptiques. L'architecture est conçue sous forme d'une grille de nœuds (des routeurs) et chaque nœud du NOC est relié soit à des mémoires (deux au maximum), soit à un élément de calcul WTAP (Winner Take All Processor). Un unique nœud est réservé pour le contrôleur (*Manager*). Le nombre de WTAPs et le nombre de mémoires dépend de la taille du GBNN mis en œuvre (nombre de clusters C et nombre de neurones par cluster L).

Dans l'architecture, le nombre de WTAPs est égal au nombre de clusters. Pour un GBNN(C,L), il y a C WTAPs. Ces modules reçoivent les connexions envoyées par les mémoires, mettent à jour le score des neurones et trient la liste contenant le score des neurones afin de déterminer le neurone (ou les neurones) ayant le meilleur score. Ainsi, chaque WTAP du NOC est associé à un unique cluster.

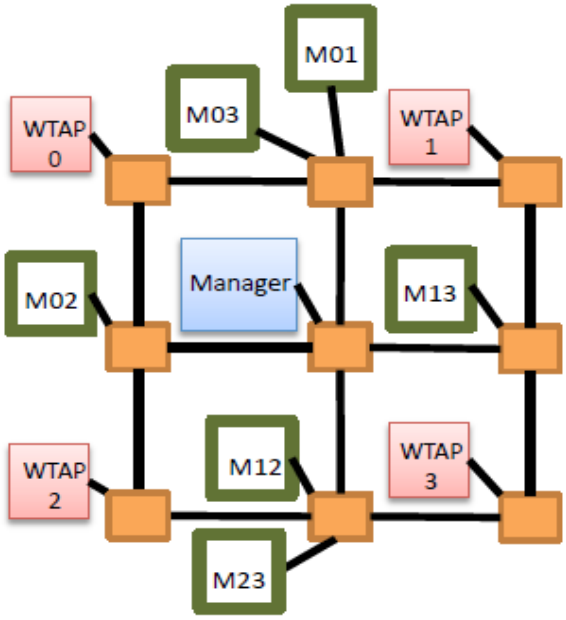


Figure IV.2 Réseau NOC mettant en oeuvre un GBNN contenant 4 clusters et étant muni de 9 nœuds, 4 nœuds pour chaque WTAP, 4 nœuds pour les mémoires et un nœud pour le manager. Image issue de [86]

Chaque mémoire contient les poids synaptiques entre les neurones de deux clusters différents. Ainsi, il y a $C * (C-1)/2$ mémoires dans l'architecture. Le nombre de bits par mots mémoire et le nombre de mots par mémoire sont égales à L.

Le rôle du manager est : d'initialiser les valeurs des neurones du réseau, de communiquer avec les mémoires à travers l'interface NOC et de contrôler les itérations de décodage

jusqu'à la détection d'un message réparé ou jusqu'à ce que le nombre maximal d'itération de décodage ait été atteint. Lorsque le manager reçoit un message à apprendre, il crée des paquets pour chaque mémoire. Ces paquets contiennent deux identifiants de neurones, un pour chaque cluster lié à la mémoire qui recevra le paquet. Cela permet de mettre à 1 le poids entre ces neurones. Lorsque l'architecture reçoit un message à décoder, le manager initialise les valeurs des neurones. Une itération de décodage se déroule en trois phases :

- Du manager aux mémoires : pour chaque cluster inconnu, le manager envoie les identifiants des neurones actifs à chaque mémoire contenant les poids synaptiques de ce cluster.
- Des mémoires aux WTAPs : après avoir reçu les informations provenant du manager, les mémoires envoient les poids synaptiques au WTAP associé au cluster effacé.
- Des WTAPs au manager : après avoir traité tous les poids synaptiques, les WTAPs ayant réalisé le décodage envoient leurs neurones actifs au manager.

Les communications entre les différents modules se font par paquets et lors de la communication entre deux modules, l'envoie de la totalité des données peut prendre plusieurs cycles (incluant la création et l'envoi des paquets).

Les architectures matérielles proposées dans l'état de l'art sont paramétrées en fonction de la taille du réseau mis en œuvre. Une fois synthétisée, elles ne permettent pas de manipuler des réseaux de différentes tailles. Cela implique qu'il n'existe aucune architecture générique mettant à œuvre les réseaux GBNN.

C. Calculs dans un réseau à clones et multiplication de matrices parcimonieuses par des vecteurs

La performance d'un GBNN dépend en partie de sa densité globale qui est équivalente à la densité de sa matrice d'adjacence. Comme les réseaux de Willshaw, les GBNNs ont de meilleures performances pour des densités faibles [55]. La matrice d'adjacence d'un GBNN ayant une bonne performance est donc *parcimonieuse*.

Définition : Matrice parcimonieuse

> Une *matrice parcimonieuse* est une matrice qui contient un pourcentage élevé d'éléments égaux à 0 (c.à.d. une matrice ayant une faible densité). Cette définition générique peut être étendu à une définition proche de celle de Wilkinson [87] : « une matrice est parcimonieuse si elle contient suffisamment de 0 pour en tirer parti ».

Définition : Multiplication d'une matrice parcimonieuse par un vecteur

> Soit W_{u*v} une matrice munie de u lignes et de v colonnes, $\boldsymbol{x_v}$ un vecteur muni de v lignes et $\boldsymbol{y_u}$ un vecteur muni de v lignes. Une *multiplication d'une matrice parcimonieuse par un vecteur* (Sparse Matrix Vector Multiplication : SpMxV) est un produit matriciel $\boldsymbol{y} = W * \boldsymbol{x}$ tel que W est une matrice parcimonieuse.

Dans un réseau à clones, le score d'un clone est égal à :

$$\boldsymbol{sc}\left(\boldsymbol{n}^{\boldsymbol{x}}_{(\boldsymbol{i},\boldsymbol{j},\boldsymbol{k})}\right)(\boldsymbol{t}+\boldsymbol{1}) = \boldsymbol{v}\left(\boldsymbol{n}^{\boldsymbol{x}}_{(\boldsymbol{i},\boldsymbol{j},\boldsymbol{k})}\right)(\boldsymbol{t}) + \sum_{\boldsymbol{j'=0,j'\neq j}}^{\boldsymbol{C-1}} \boldsymbol{max}\left(\sum_{n^{x'}_{(i',j',k)} \in N(j',k)} \left(\boldsymbol{w}\left(\boldsymbol{n}^{\boldsymbol{x}}_{(\boldsymbol{i},\boldsymbol{j},\boldsymbol{k})};\ \boldsymbol{n}^{\boldsymbol{x'}}_{(\boldsymbol{i'},\boldsymbol{j'},\boldsymbol{k})}\right) * v\left(n^{x'}_{(i',j',k)}\right)(t)\right); \boldsymbol{1} \right) \quad (44)$$

Si $\boldsymbol{Sc}(\boldsymbol{t}+\boldsymbol{1})$ est un vecteur contenant le score de tous les clones du réseau lors d'une itération $t+1$ et si $\boldsymbol{v}(\boldsymbol{t})$est un vecteur contenant les états de tous les clones du réseau pour l'itération t, alors $\boldsymbol{Sc}(\boldsymbol{t}+\boldsymbol{1})$ peut être représenté par l'expression suivante :

$$\boldsymbol{Sc}(\boldsymbol{t}+\boldsymbol{1}) = \boldsymbol{v}(\boldsymbol{t}) + W \boxtimes \boldsymbol{v}(\boldsymbol{t}) \quad (45)$$

Avec W étant la matrice d'adjacence du réseau.

Définition : Contribution d'un cluster distant pour un clone

> La *contribution d'un cluster distant* $c_{(j',k)}$ *pour un clone* $v(n^x_{(i,j,k)})$ est la valeur apportée au score du clone $v(n^x_{(i,j,k)})$ par ce cluster distant. Elle est donnée par l'expression $\bigvee_{n^{x'}_{(i',j',k)} \in N(j',k),}(\ )$ présente dans l'équation (44).

L'expression $W \boxtimes \boldsymbol{v}(\boldsymbol{t})$ s'apparente à une multiplication d'une matrice parcimonieuse par un vecteur. Cependant, elle comporte quelques différences par rapport à ce type de calcul. En effet, pour un score $sc\left(n^x_{(i,j,k)}\right)(t+1)$, certaines parties de la somme sont majorées à 1 (OU logique présent dans l'expression $\bigvee_{n^{x'}_{(i',j',k)} \in N(j',k),}(\ )$ de l'équation (44)).

Dans le cadre des réseaux à clones, le vecteur $\boldsymbol{v}(\boldsymbol{t})$ est un vecteur binaire et parcimonieux car seul un faible nombre des clones du réseau ont une valeur égale à 1. Ainsi, le vecteur résultat $\boldsymbol{Sc}(\boldsymbol{t}+\boldsymbol{1})$ est lui aussi parcimonieux. En outre, la multiplication s'assimile à un *ET* logique $\left(v\left(n^{x'}_{(i',j',k)}\right)(t) \wedge w\left(n^{x}_{(i,j,k)};\ n^{x'}_{(i',j',k)}\right)\right)$ car il s'agit de la multiplication d'un bit par un autre bit.

Il est important de noter que dans le décodage d'un message, d'autres opérations sont à prendre en compte après le calcul du score des clones. En effet, ce calcul des scores est suivi d'une opération de WTA qui consiste en deux phases :

- Calcul du score maximal du réseau ;

- Mise à 1 des valeurs des clones ayant un score égal au score maximal et mise à 0 des valeurs des clones ayant un score inférieur à ce score maximal.

Plusieurs implémentations ont été proposées pour mettre en œuvre des SpMxV que ce soit sur CPU (Central Processing Unit) [88], FPGA [89]–[92] ou encore sur GPU (Graphics Processing Unit) [93]–[95]. Notre proposition est de s'inspirer de ces travaux pour mettre en œuvre des systèmes performants et génériques réalisant le décodage des réseaux à clones. Dans les sections suivantes, les principaux modes de représentation des matrices parcimonieuses ainsi qu'une architecture performante mettant en œuvre une multiplication de matrice parcimonieuse par un vecteur sont présentés.

D. Modes de représentation des matrices parcimonieuses

Les matrices parcimonieuses interviennent dans de nombreux domaines : réseaux de neurones [55], ingénierie industrielle [96], traitement d'image [97] (etc.). De nombreuses études portant sur les architectures permettant de réaliser des produits de matrices parcimonieuses par des vecteurs ont été réalisées et se focalisent sur trois problèmes principaux [98] :

- Le mode de représentation qui doit être le moins coûteux possible en termes de ressources mémoires et le moins complexe en terme de temps.
- La performance qui doit être la plus élevée possible (temps de calcul) par rapport aux ressources matérielles utilisées
- L'efficacité énergétique qui est la performance par rapport à l'énergie consommée.

Il existe quatre principaux modes de représentation des matrices parcimonieuses [92], [98] :

- Par matrice d'adjacence (*Bit-Vector :* BV) ;
- Par coordonnées (*COOrdinate format :* COO) ;
- Par liste de colonnes pour chaque ligne (*Compressed Sparse Row :* CSR) ;
- Par liste de lignes pour chaque colonne (*Compressed Sparse Column :* CSC).

Notations :	
W_{u*v}	Matrice parcimonieuse contenant u lignes et v colonnes. Son nombre total d'éléments est $\|W\| = u * v$
$d(W)$	Densité de W. Ainsi, le nombre d'éléments non nuls dans la matrice est $d(W) * \|W\|$
$Nbc(W)$	Nombre de bits de codage des éléments de W
$Nbv(e)$	Nombre de bits requis pour coder un élément $e \in \mathbb{R}$

Dans les sous-sections suivantes, chaque mode de représentation est présenté en indiquant d'une part le coût de mémorisation en bits de la matrice en utilisant ce mode et d'autre part sa *complexité temporelle* (nombre de cycles requis pour lire tous les éléments de la matrice selon ce mode).

1. BV

a) Définition

Une matrice représentée à l'aide d'une matrice d'adjacence BV utilise les deux vecteurs suivants :

- Un vecteur *bit-vector* qui représente la matrice mais linéarisée et sous forme binaire. Chaque élément de ce vecteur indique si la position considérée contient une valeur nulle (0) ou non (1), c.à.d. :

$$bit - vector(i * v + j) = \begin{cases} 1 \; si \; W(i,j) \neq 0 \\ 0, sinon \end{cases}$$

- Un vecteur *données* qui contient les valeurs non nulles présentes à ces positions.

Lorsque les éléments de la matrice sont binaires, le vecteur *données* peut être supprimé car chaque bit du vecteur *bit-vector* indique la présence d'un élément non nul et cet élément ne peut avoir que 1. Cette représentation est celle qui est utilisées dans la Figure II.11 montrant les poids synaptiques d'un GBNN. C'est aussi cette représentation qui est utilisée dans toutes les architectures matérielles spécialisées mettant en œuvre les GBNNs. Ces architectures réalisent alors un stockage intégral de la matrice.

Exemple :

Matrice	Représentation
$\begin{bmatrix} 1 & 0 & 4 & 0 \\ 3 & 7 & 0 & 0 \\ 0 & 0 & 2 & 9 \\ 5 & 8 & 0 & 7 \end{bmatrix}$	$bit - vector = [1\,0\,1\,0\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,1]$ $données = [1\,4\,3\,7\,2\,9\,5\,8\,7]$

b) Propriétés

Le vecteur *bit-vector* possède un nombre d'éléments égal à celui de la matrice : $|bit - vector| = |W|$. Ainsi, la complexité du mode BV est :

$$Cx(M) = |W| \tag{46}$$

Etant donné que chaque élément de ce vecteur est un bit alors son coût de mémorisation est de $|W|$ bits. De plus, le coût de mémorisation du vecteur *données* est de $d(W) * |W| *$

$Nbc(W)$ bits (nombre d'éléments de la matrice multiplié par le nombre de bits pour coder chaque élément).

Le coût total de W en mode BV est alors :

$$Ct(M) = |W| + d(W) * |W| * Nbc(W) \text{ bits} \quad (47)$$

2. COO

a) Définition

Une matrice représentée à l'aide de coordonnées *COO* utilise les trois vecteurs suivants : *lignes*, *colonnes* et *données*.

Dans le vecteur *lignes*, chaque élément représente la ligne où se trouve une valeur non nulle. Chaque élément de *lignes* est associé à un unique élément du vecteur *colonnes* qui indique la colonne où se trouve cette valeur.

Le vecteur *données* contient la valeur de l'élément présent à ces coordonnées dans la matrice.

Exemple :

Matrice	Représentation
$\begin{bmatrix}1 & 0 & 4 & 0\\3 & 7 & 0 & 0\\0 & 0 & 2 & 9\\5 & 8 & 0 & 7\end{bmatrix}$	$lignes = [0\ 0\ 1\ 1\ 2\ 2\ 3\ 3\ 3]$ $colonnes = [0\ 2\ 0\ 1\ 2\ 3\ 0\ 1\ 3]$ $données = [1\ 4\ 3\ 7\ 2\ 9\ 5\ 8\ 7]$

b) Propriétés

Le vecteur *lignes* et le vecteur *colonnes* possèdent tous les deux : $d(W) * |W|$ éléments. Ainsi, la complexité du mode *COO* est :

$$Cx(W) = d(W) * |W| \quad (48)$$

Chaque élément de *lignes* doit pouvoir pointer sur n'importe quelle ligne de *W*. Il doit donc être codé sur $Nbv(u)$ bits. Le coût de mémorisation de *lignes* est donc de $d(W) * |W| * Nbv(u)$ bits. D'un autre côté chaque élément de *colonnes* doit pouvoir pointer n'importe quelle colonne de *M*. Il doit donc être codé sur $Nbv(v)$ bits. Le coût de mémorisation de *colonnes* est donc de $d(W) * |W| * Nbv(v)$ bits. Le coût de *données* est inchangé.

Le coût total de *W* en mode *COO* est alors de :

$$Ct(W) = d(W) * |W| * Nbv(u) + d(W) * |W| * Nbv(v) + d(W) * |W| * Nbc(W) \text{ bits} \quad (49)$$

3. CSR

a) Définition

Une matrice représentée à l'aide d'une liste de colonnes pour chaque ligne *CSR* utilise les trois vecteurs suivants : *pointeurs*, *colonnes* et *données*.

L'usage des vecteurs *données* et *colonnes* est inchangé par rapport à COO. Concernant le vecteur *indices*, chaque élément i (sauf le dernier) indique pour une ligne précise i l'emplacement dans le vecteur *colonnes* où commence sa liste des colonnes contenant des éléments non nuls. Le dernier élément de *pointeurs* indique le nombre total d'éléments non nuls dans la matrice.

Exemple :

Matrice	Représentation
$\begin{bmatrix}1 & 0 & 4 & 0\\ 3 & 7 & 0 & 0\\ 0 & 0 & 2 & 9\\ 5 & 8 & 0 & 7\end{bmatrix}$	$pointeurs = [0\ 2\ 4\ 6\ 9]$ $colonnes = [0\ 2\ 0\ 1\ 2\ 3\ 0\ 1\ 3]$ $données = [1\ 4\ 3\ 7\ 2\ 9\ 5\ 8\ 7]$

b) Propriétés

Le vecteur *colonnes* contient $d(W) * |W|$ éléments tandis que le vecteur *pointeurs* contient $u + 1$ éléments pour chaque ligne de W. Ainsi, la complexité en mode CSR est identique à celle du mode COO, c.à.d. $d(W) * |W|$.

Chaque élément de *pointeurs* doit pouvoir pointer sur chaque élément de *W*. Leur valeur maximale est donc de $|W|$ ce qui implique un codage des éléments de *pointeurs* sur $Nbv(|W|)\ bits$. Ainsi, le coût de mémorisation de ce vecteur est de : $Nbv(|W|) * (u + 1)$ bits. Le coût de *données* est inchangé.

Le coût total de *W* en mode *CSR* est alors de :

$$Ct(M) = (u + 1) * Nbv(|W|) + d(W) * |W| * Nbv(v) + d(W) * |W| * Nbc(W)\text{bits} \qquad (50)$$

4. CSC

a) Définition

Une matrice représentée à l'aide d'une liste de lignes pour chaque colonne CSC utilise les trois vecteurs suivants : *pointeurs*, *lignes* et *données*.

L'usage des vecteurs *données* et *lignes* est inchangé par rapport à COO. Concernant le vecteur *indices*, chaque élément j indique pour une colonne précise j (sauf le dernier élément) l'emplacement dans le vecteur *lignes* où commence sa liste de lignes contenant les éléments non nuls. Le dernier élément indique le nombre d'éléments non nuls dans la matrice.

Le mode CSC est très proche du mode CSR car la représentation par liste de lignes du CSC est similaire la représentation par liste de colonnes du CSR.

Exemple :

Matrice	Représentation
$\begin{bmatrix}1 & 0 & 4 & 0\\3 & 7 & 0 & 0\\0 & 0 & 2 & 9\\5 & 8 & 0 & 7\end{bmatrix}$	$pointeurs = [0\ 3\ 5\ 7\ 9]$ $lignes = [0\ 1\ 3\ 1\ 3\ 0\ 2\ 2\ 3]$ $données = [1\ 3\ 5\ 7\ 8\ 4\ 2\ 9\ 7]$

b) Propriétés

Le vecteur *lignes* contient $d(W) * |W|$ éléments tandis que le vecteur *pointeurs* contient $v + 1$ éléments pour chaque colonne de W. Ainsi la complexité en mode CSC est identique à celle du mode COO, c.à.d. $d(W) * |W|$.

Chaque élément de *pointeurs* doit pouvoir pointer sur chaque élément de W. Leur valeur maximale est donc de $|W|$ ce qui implique un codage sur $Nbv(|W|)$ bits. Ainsi, le coût de mémorisation de ce vecteur est de : $(v + 1) * Nbv(|W|)\ bits$.

Le coût total de W en mode CSC est alors de :

$$Ct(W) = (v + 1) * Nbv(|W|) + d(W) * |W| * Nbv(u) + d(W) * |W| * Nbc(W) \text{ bits} \qquad (51)$$

5. Exemple

Le Tableau 4 montre la matrice du GBNN Figure II.11 représentée selon les différents modes de représentation exposés.

Tableau 4 Représentation de la matrice d'un GBNN selon différents modes

Modes de représentation	Représentation
Matrice de base	$\begin{bmatrix}0&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&1&0&0\\0&0&0&0&1&1&1&0&0\\0&0&0&0&0&0&1&0&0\\1&0&0&0&0&0&1&0&0\\0&1&1&0&0&0&1&0&0\\0&1&1&1&1&1&0&0&0\\0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0\end{bmatrix}$
BV	$bit - vector =$ $\begin{bmatrix}0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\\0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1\,0\,0\\0\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\end{bmatrix}$
COO	$lignes = [1\ 1\ 2\ 2\ 2\ 3\ 4\ 4\ 5\ 5\ 5\ 6\ 6\ 6\ 6\ 6\ 7]$ $colonnes = [3\ 6\ 4\ 5\ 6\ 6\ 0\ 6\ 1\ 2\ 6\ 1\ 2\ 3\ 4\ 5\ 0]$
CSR	$pointeurs = [0\ 0\ 2\ 5\ 6\ 8\ 11\ 16\ 17\ 18\ ]$

	$colonnes = [3\ 6\ 4\ 5\ 6\ 6\ 0\ 6\ 1\ 2\ 6\ 1\ 2\ 3\ 4\ 5\ 0]$
CSC	$pointeurs = [0\ 0\ 2\ 5\ 6\ 8\ 11\ 16\ 17\ 18\ ]$ $lignes = [1\ 1\ 2\ 2\ 2\ 3\ 4\ 4\ 5\ 5\ 5\ 6\ 6\ 6\ 6\ 6\ 7]$

E. Architecture matérielle générique mettant en œuvre la multiplication d'une matrice parcimonieuse par un vecteur

L'architecture présentée dans cette section a été proposée par Dorrance et al. [92]. Cette architecture est plus performante que les mises en œuvre des SpMxVs sur différentes plateformes (GPU, FPGA, ordinateurs multiprocesseurs). Elle possède aussi une meilleure efficacité énergétique et est capable de passer à l'échelle. Sa seule limitation est le débit des mémoires qui alimentent l'architecture en données.

Cette architecture se base une représentation de la matrice en CSC. Selon ce mode de représentation, le calcul de $\boldsymbol{y} = W * \boldsymbol{x}$ se déroule sur plusieurs itérations et fonctionne selon le principe suivant. A chaque itération, des valeurs partielles $\boldsymbol{y}(:)$ de toutes les lignes de $\boldsymbol{y}$ sont calculées simultanément en multipliant un unique élément $\boldsymbol{x}(ligX)$ de $\boldsymbol{x}$, par une unique colonne $W(:, ligX)$ de W. Le résultat de ce calcul est un vecteur qui est ensuite cumulé à la valeur actuelle de $\boldsymbol{y}$. Ainsi, $\boldsymbol{y}(:)(t+1)$, la valeur actuelle de $\boldsymbol{y}$ après *t+1* itérations est donnée par :

$$\boldsymbol{y}(:)(t+1) = W(:, ligX) * \boldsymbol{x}(ligX) + \boldsymbol{y}(:)(t) \tag{52}$$

Avec $\boldsymbol{y}(:)(0) = 0$ (Vecteur nul)

La Figure IV.3 illustre la multiplication d'une matrice W par un vecteur x. Le vecteur résultat $\boldsymbol{y}$ est donné par la somme de calculs comportant deux éléments : un vecteur étant une colonne $W(:, ligX)$ de W et un unique élément du vecteur x. Dans cette figure, les cases blanches représentent la valeur 0 et les autres cases représentent des valeurs non nulles. La première ligne de $\boldsymbol{y}$, $\boldsymbol{y}(0)$ est obtenue en cumulant successivement les calculs. Ainsi, $\boldsymbol{y}(0) = W(0,0) * \boldsymbol{x}(0) + W(0,1) * \boldsymbol{x}(1) + W(0,2) * \boldsymbol{x}(2) + \cdots + W(0,7) * \boldsymbol{x}(7)$.

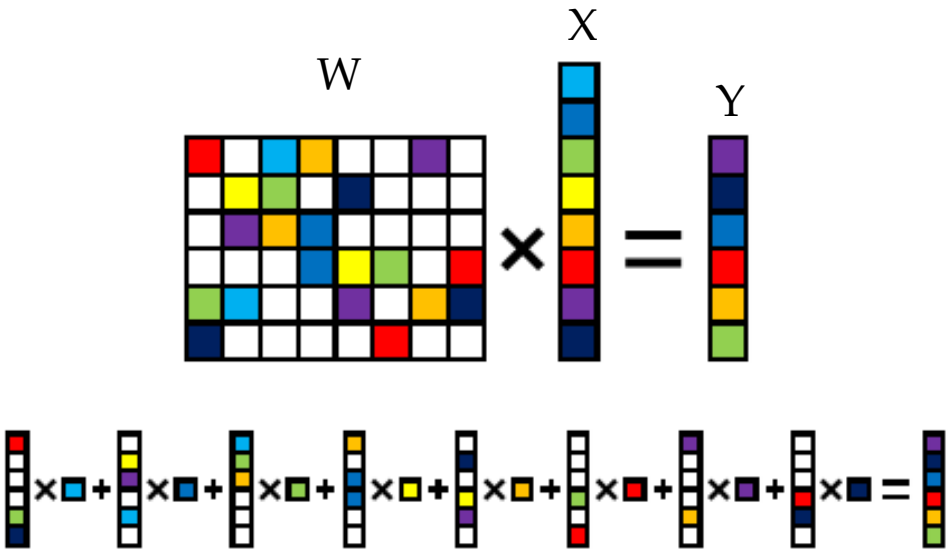


Figure IV.3 Multiplication en CSC **d'une matrice parcimonieuse par un vecteur. Image issue de [92]**

1. Composants de l'architecture

L'architecture proposée par Dorrance et al. comporte cinq modules :

- Une mémoire DRAM (Dynamic Random Access Memory) externe au système et qui contient le vecteur *pointeurs*, le vecteur *lignes* et le vecteur *données* faisant partie de la représentation en CSC de la matrice parcimonieuse *W*.
- Deux mémoires SRAM (Static Random Access Memory), QDR0 et QDR1 externes au système qui sont dédiées respectivement au vecteur d'entrée *X* et au vecteur de sortie Y. Le terme QDR fait référence aux mémoires QDRII+SRAM de Cypress [99] ;
- Un contrôleur mémoire pour gérer les accès aux mémoires DRAM, QDR0 et QDR1.
- Le contrôleur SpMxV qui contrôle la totalité des calculs et la communication avec les mémoires.
- Les éléments de calcul (Processing Element : *PE*) qui reçoivent des données du calcul (éléments de la matrice, éléments du vecteur, numéro de la ligne où va être stocké le résultat…) et réalisent le calcul (multiplication entre les entrées et addition avec l'ancienne valeur).

La Figure IV.4 montre un aperçu de cette architecture.

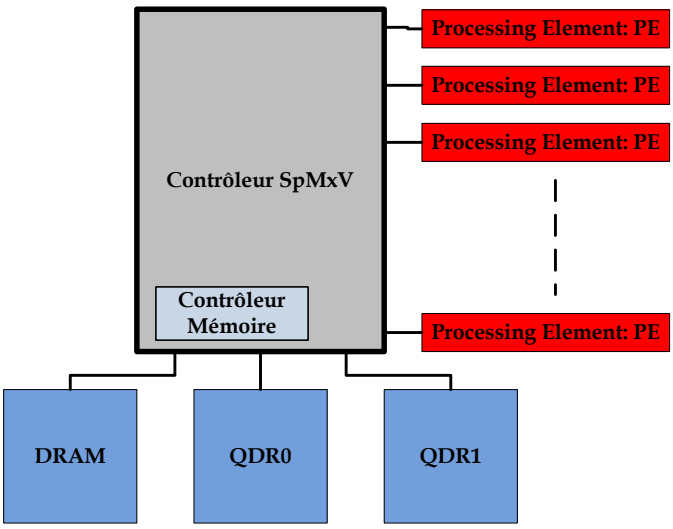


Figure IV.4 Schéma d'une architecture réalisant un SpMxV [92]

Chaque *PE* réalise le produit d'un unique élément $W(lixW, ligX)$ de W avec un unique élément $\boldsymbol{x}(ligX)$ de $\boldsymbol{x}$ et additionne le résultat à $\boldsymbol{y}(ligW)$.

A chaque *PE* est attribué un ensemble de lignes de la matrice W et un ensemble de lignes de y de telle sorte que si un élément de $W(ligW, ligX)$ est lu à partir de DRAM, il est transmis automatiquement vers son *PE* prédéfini en fonction de la ligne $ligW$ à laquelle il appartient. Si l'architecture contient une liste de *PEs*, $\mathcal{L}_{PE} = (PE_i)_{0 \leq i \leq |PE|-1}$, une ligne $ligW$ sera associé à $PE_{ligW\ MOD\ |PE|}$. Ainsi, si $|PE| = 4$, les lignes 2, 4, 6, 7 de la matrice seront associées respectivement aux *PEs* : PE_2, PE_0, PE_2, PE_3. D'un autre côté, chaque *PE* contient une partie de $\boldsymbol{y}$ de telle sorte que la valeur $\boldsymbol{y}(ligY)$ d'une ligne $ligY$ de $\boldsymbol{y}$ sera contenue dans le *PE* défini par $PE_{ligY\ MOD\ |PE|}$.

Chaque *PE* contient :

- Un module multiplieur-accumulateur (Multiplier-ACcumulator: MAC) permettant de calculer le produit d'un unique élément provenant de W et d'un unique élément provenant de $\boldsymbol{x}$, avant de l'accumuler avec la valeur d'une ligne de $\boldsymbol{y}$.
- Une mémoire interne qui contient les valeurs de chaque ligne de $\boldsymbol{y}$ associées à ce *PE*.

La Figure IV.5 montre l'aperçu d'un élément de calcul.

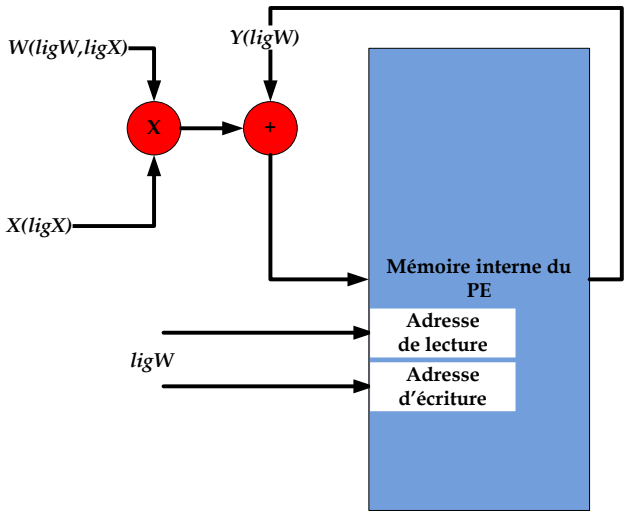


Figure IV.5 Aperçu d'un *PE*. Image inspirée de [92]

2. Fonctionnement de l'architecture

A l'initialisation du calcul, le vecteur $\boldsymbol{x}$ est situé dans la mémoire externe QDR0 tandis que la matrice W est stockée dans la mémoire DRAM. La mémoire interne de chaque *PE* est initialisée par des 0. $\boldsymbol{x}$ et W sont chargés au fur et à mesure du calcul par le contrôleur mémoire ce qui permet d'alimenter constamment l'architecture en données. Le chargement de W s'effectue par groupe de colonnes tandis que le chargement de $\boldsymbol{x}$ s'effectue par groupe de lignes.

L'architecture fonctionne alors selon le principe suivant :

1. Pendant le calcul, un élément $\boldsymbol{x}(ligX)$ de $\boldsymbol{x}$ est lu. Cela implique que la colonne $W(:, ligX)$ de W est sélectionnée, doit être lue et que ses éléments non nuls doivent être traités élément après élément.
2. Ensuite, une unique ligne $ligW$ de W associée à la colonne $\boldsymbol{x}(ligX)$ est lue. L'élément $W(ligW, ligX)$ de la matrice est aussi lu. L'ensemble est ensuite redirigé vers le *PE* attribué à la ligne $ligW$.
3. Lorsque le *PE* reçoit $ligW$, $W(ligW, ligX)$ et $\boldsymbol{x}(ligX)$, il lit la valeur actuelle de $\boldsymbol{y}(ligW)$ dans la mémoire interne du *PE*. Ensuite, il réalise le calcul $W(ligW, ligX) * \boldsymbol{x}(ligX)$, additionne avec la valeur $\boldsymbol{y}(ligW)$ lue à partir de la mémoire interne. Enfin, il sauvegarde le nouveau résultat dans cette même mémoire interne et à l'adresse où est stockée $\boldsymbol{y}(ligW)$.
4. Pendant ce temps, une autre ligne $ligW2$ et la donnée $W(ligW2, ligX)$ peuvent être lues. Ces données seront aussi transférées au *PE* attribué à $ligW2$. Comme précédemment, le *PE* va réaliser le calcul et mettre à jour la valeur de la ligne correspondante à $\boldsymbol{y}(ligW2)$.

5. A la fin du parcours de la colonne $W(:, ligX)$ de W, la valeur suivante de $\boldsymbol{x}$, $\boldsymbol{x}(ligX + 1)$ est sélectionnée et le processus recommence avec la nouvelle colonne $W(:, ligX + 1)$. (Retour à 2)
6. Le calcul s'achève lorsque tous les éléments de $\boldsymbol{x}$ ont été traités. Le résultat, $\boldsymbol{y}$ est alors contenu et distribué dans les mémoires internes des *PEs*.

3. Avantages et inconvénients

Cette architecture possède les avantages suivants :

- Les différents *PEs* sont indépendants et n'ont pas besoin de synchroniser leurs calculs. (Pas de communication entre les *PEs*)
- L'attribution des données et des calculs aux *PEs* est rapide car simple et prédéfinie à l'avance. Chaque donnée est automatiquement routée vers son *PE* associé. Cela implique que le module utilisé pour router les données ($W(ligW; ligX)$ et $\boldsymbol{x}(ligX)$) est peu complexe.
- Un nombre de *PEs* réduit est capable de consommer tout le débit de la mémoire. Dans les évaluations, le nombre maximal de *PEs* pouvant être utilisé (jusqu'à saturation de la plateforme FPGA) est de 96 mais 64 seulement ont été nécessaires pour consommer toute la bande passante des mémoires.
- Le contrôleur mémoire charge les données de manière séquentielle ce qui implique qu'il est peu complexe. Par exemple, il charge la matrice colonne par colonne.

Cependant, cette architecture nécessite que les mémoires internes des *PEs* doivent pouvoir contenir tous les éléments de $\boldsymbol{y}$. Dans le cas contraire, des opérations d'écriture et de chargement avec une mémoire externe sont nécessaires, ce qui est coûteux en temps de calcul.

F. Bilan

Dans ce chapitre, l'état de l'art sur les architectures mettant en œuvre les réseaux GBNN a été présenté tandis que les modes de stockage et les calculs existants au sein de ces réseaux ont été analysés en détail.

Toutes les architectures existantes sont spécialisées c.à.d. qu'une fois synthétisée, elles permettent de ne manipuler que des réseaux d'une taille précise. De plus, la plupart d'entre elles utilisent des calculs et des communications qui s'exécutent en parallèle.

Ainsi, le problème de l'absence d'architectures matérielles génériques a été soulevé. De plus il n'existe aucune architecture mettant en œuvre les réseaux à clones. Après avoir montré que le décodage au sein des réseaux à clones avait des similitudes avec des multiplications de matrices parcimonieuses par des vecteurs (SpMxVs), différents modes de représentation

de matrices parcimonieuses ont été présentés. De plus, une architecture performante mettant en œuvre ce type de calcul a été explorée.

Dans la suite de ce document, d'une part, des simplifications du modèle GBNN originel et des architectures spécialisées mettant en œuvre des réseaux GBNNs utilisant moins de ressources que les architectures de l'état de l'art sont présentées. Ces propositions ont été proposées avant l'architecture NOC et ne se compare pas à cette dernière. D'autre part, une architecture générique qui une fois synthétisée, permet de manipuler des réseaux à clones de tailles variables, est proposée. Cette architecture s'inspire de celle qui a été présentée dans ce chapitre et qui réalisent des SpMxVs.

CHAPITRE V. Modèle et Architectures Optimisés de Réseaux à Clusters Binaires

Sommaire du Chapitre :

Dans ce chapitre, le modèle GBNN est simplifié et des améliorations permettant de réduire le coût en ressources des architectures matérielles mettant en œuvre ce modèle sont proposées.

A. Introduction

Les architectures proposées dans [64] mettent œuvre des calculs et des communications qui se font en parallèle. Cela signifie qu'à chaque cycle, tous les neurones reçoivent la totalité des données (poids et valeurs de neurones) dont ils ont besoin pour réaliser leurs calculs. De plus, ces architectures stockent intégralement la matrice de poids synaptiques du réseau sans tenir compte de sa symétrie.

Tandis que la première propriété conduit à une architecture rapide mais utilisant beaucoup de ressources de calcul, la seconde conduit à une architecture coûteuse en ressources de stockage.

L'ensemble de ces inconvénients limite la taille maximale des architectures qui peuvent être mises en œuvre pour la même plateforme comme nous le verrons dans les résultats présentés dans la section dédiée aux expériences.

Afin d'augmenter cette taille maximale, des optimisations sont nécessaires. Dans ce chapitre, les optimisations du modèle GBNN originel et des architectures sont introduites progressivement suivant trois axes: calcul, mémoire et communication. Dans le but de diminuer la complexité des processus de restitution du message, le modèle GBNN traditionnel est transformé en un modèle totalement binaire. Cette modification permet de simplifier le processus de calcul du score et d'éliminer l'étape de Winner Take All (WTA). Afin de réduire à la fois le nombre d'éléments de stockage et la complexité du processus d'apprentissage, nous proposons de mémoriser uniquement la moitié des poids synaptiques car dans un GBNN, les connexions sont symétriques. Finalement, afin de supprimer les goulets d'étranglement se situant dans les calculs et dans les poids synaptiques, nous sérialisons l'architecture du réseau. Toutes les améliorations proposées dans cette partie visent à optimiser les performances en surface et en temps de l'architecture matérielle tout en conservant la fonctionnalité du modèle GBNN originel. Ces améliorations s'inspirent des travaux de Chavet et al. [100].

B. Calcul totalement binaire

Les modèles proposés par Gripon et Berrou [62] reposent sur des connexions binaires entre neurones de différents clusters. Ces connexions sont utilisées dans l'opération de décodage afin de calculer le score des neurones et de réaliser un algorithme de WTA (Voir CHAPITRE II.C.5.a)(3)). Nous proposons de remplacer toutes les opérations arithmétiques entières par des équations logiques. Cela revient à définir un modèle de réseau de neurones totalement binaire (c.à.d. à poids binaires et à calculs binaires). Cette propriété qui a été proposée au début par Chavet et al. [100] et qui a été partiellement utilisée pour concevoir des mémoires

associatives par Jarohalli et al. [64] est détaillée dans cette section. Le modèle binaire proposé permet de se débarrasser de l'étape de WTA et d'obtenir les mêmes performances que le modèle GBNN amélioré [78].

Durant la phase d'apprentissage, chaque nouveau message à apprendre conduit à l'enregistrement d'une nouvelle clique. Comme dans le modèle originel, mémoriser une clique dans un réseau totalement binaire revient à stocker les poids des connexions entre les neurones appartenant à la clique (c.à.d. mettre les valeurs à 1 dans les éléments de mémorisation correspondants). Cependant, la récupération des messages décrite dans [62], peut être modifiée pour prendre en compte ce fait : un neurone ne peut être actif que s'il appartient à une clique potentielle, c.à.d. s'il est connecté à au moins un neurone actif dans chaque cluster connu.

Le principe de base est celui du *vote unanime* : un neurone $n_{(i,j)}$ est actif dans un cluster donné j (c.à.d. $v\left(n_{(i,j)}\right) = 1$) si au moins un des neurones actifs dans chaque cluster distant actif (c.à.d. *neurone distant actif*) indique qu'il est connecté à $n_{(i,j)}$. Cela signifie que dans la phase d'apprentissage, une connexion avec ce neurone a été stockée pour chacun de ses neurones distants.

Cette approche modifie la conception du module de décodage. Au lieu de réaliser un WTA basé sur des valeurs entières, il est possible d'utiliser des équations booléennes pour calculer les valeurs de chaque neurone sans perte de généralité. Ainsi, les équations (23) et (24) deviennent inutiles et peuvent être remplacées par l'équation booléenne (53). Le vote unanime est représenté par l'usage de la conjonction de tous les votes ($\bigwedge_{j'=0,j'\neq j}^{C-1}(\dots)$).

$$v\left(n_{(i,j)}\right)(t+1) = \bigwedge_{j'=0,j'\neq j}^{C-1}\left(\bigvee_{i'=0}^{L-1}\left(w\left(n_{(i,j)};n_{(i',j')}\right)\wedge v\left(n_{(i',j')}\right)(t)\right)\vee \dots\right) \tag{53}$$

Dans l'équation (53), $v\left(n_{(i,j)}\right)(t+1)$ représente la valeur du neurone $n_{(i,j)}$ à l'instant $t+1$. Elle est obtenue par la conjonction logique des valeurs de tous les autres neurones actifs $n_{(i',j')}$, $(j' \neq j)$ connectés à $n_{(i,j)}$ ($w\left(n_{(i,j)};n_{(i',j')}\right)$ pour tous les clusters distants ($0 \leq j' \leq C$ et $j' \neq j$). Ainsi, des éléments doivent être ajoutés pour ignorer les clusters non actifs. En effet, si jamais un cluster distant donné est inactif (la partie correspondante du message est manquante), sa participation au vote doit être neutralisée pour éliminer des effets indésirables. Cet élément est appelé *transparence d'un cluster* (Voir (54)) :

$$v\left(n_{(i,j)}\right)(t+1) = \bigwedge_{j'=0,j'\neq j}^{C-1}\left(\bigvee_{i'=0}^{L-1}\left(w\left(n_{(i,j)};n_{(i',j')}\right)\wedge v\left(n_{(i',j')}\right)(t)\right)\vee\overline{\bigvee_{i'=0}^{L-1}v\left(n_{(i',j')}\right)(t)}\right) \tag{54}$$

Cette dernière contrainte est exprimée par le dernier terme de l'expression dans (54). On peut remarquer que toutes les sommes et produits de la formule originale ont été retirés. De plus, le WTA disparaît explicitement (comparaison et sélection) parce que chaque neurone le réalise implicitement (vote).

Par exemple, considérons le GBNN présenté dans la Figure II.10. Celui-ci est formalisé en tant que réseau totalement binaire. La matrice synaptique (Voir Figure II.11) reste inchangée. Si un message partiel [?,1,0] est présenté au réseau, celui-ci doit prendre une décision en déterminant quel est le neurone associé à la valeur du premier symbole. Au début du décodage, les valeurs des neurones connues sont mises à 1 tandis que les valeurs des autres sont mises à 0. Alors, pour un cluster actif, (ex : cluster $\mathcal{C}_2$), la transparence associée sera égale à 0 tandis que les clusters inactifs (ex : cluster $\mathcal{C}_0$) auront une transparence égale à 1 et n'impacteront pas l'évaluation dans les autres clusters inactifs.

$$\overline{v(n_{(0,0)})(0) \vee v(n_{(1,0)})(0) \vee v(n_{(2,0)})(0)} = \overline{0 \vee 0 \vee 0} = 1$$

Dans un cluster inactif (ex : $\mathcal{C}_0$), l'équation (54) permet de déterminer quels neurones doivent être activés. Dans notre exemple, étant donné que tous les clusters distants du cluster $\mathcal{C}_0$ sont actifs, alors leur bit de transparence sera égal à 0.

Dans notre exemple, l'équation (54) est appliquée à tous les neurones du cluster inactif $\mathcal{C}_0$.

Ainsi, nous obtenons :

$$\begin{aligned} v(n_{(0,0)})(1) = &\Big(\big(v(n_{(0,1)})(0) \wedge w(n_{(0,0)}; n_{(0,1)})\big) \vee (v(n_{(1,1)})(0) \wedge w(n_{(0,0)}; n_{(0,1)})) \\ &\vee \big(v(n_{(2,1)})(0) \wedge w(n_{(0,0)}; n_{(0,1)})\big) \vee 0\Big) \\ &\wedge \Big(\big(v(n_{(0,2)})(0) \wedge w(n_{(0,0)}; n_{(0,2)})\big) \vee (v(n_{(1,2)})(0) \wedge w(n_{(0,0)}; n_{(0,2)})) \\ &\vee \big(v(n_{(2,2)})(0) \wedge w(n_{(0,0)}; n_{(0,2)})\big) \vee 0\Big) \\ v(n_{(0,0)})(1) = &\Big(((0 \wedge 0) \vee (1 \wedge 0) \vee (0 \wedge 0)) \vee 0\Big) \wedge \Big(((1 \wedge 0) \vee (0 \wedge 0) \vee (0 \wedge 0)) \vee 0\Big) = 0 \end{aligned}$$

$$\begin{aligned} v(n_{(1,0)})(1) = &\Big(\big(v(n_{(0,1)})(0) \wedge w(n_{(1,0)}; n_{(0,1)})\big) \vee (v(n_{(1,1)})(0) \wedge w(n_{(1,0)}; n_{(0,1)})) \\ &\vee \big(v(n_{(2,1)})(0) \wedge w(n_{(1,0)}; n_{(0,1)})\big) \vee 0\Big) \\ &\wedge \Big(\big(v(n_{(0,2)})(0) \wedge w(n_{(1,0)}; n_{(0,2)})\big) \vee (v(n_{(1,2)})(0) \wedge w(n_{(1,0)}; n_{(0,2)})) \\ &\vee \big(v(n_{(2,2)})(0) \wedge w(n_{(1,0)}; n_{(0,2)})\big) \vee 0\Big) \\ v(n_{(1,0)})(1) = &\Big(((0 \wedge 0) \vee (1 \wedge 0) \vee (0 \wedge 0)) \vee 0\Big) \wedge \Big(((1 \wedge 0) \vee (0 \wedge 0) \vee (0 \wedge 0)) \vee 0\Big) = 0 \end{aligned}$$

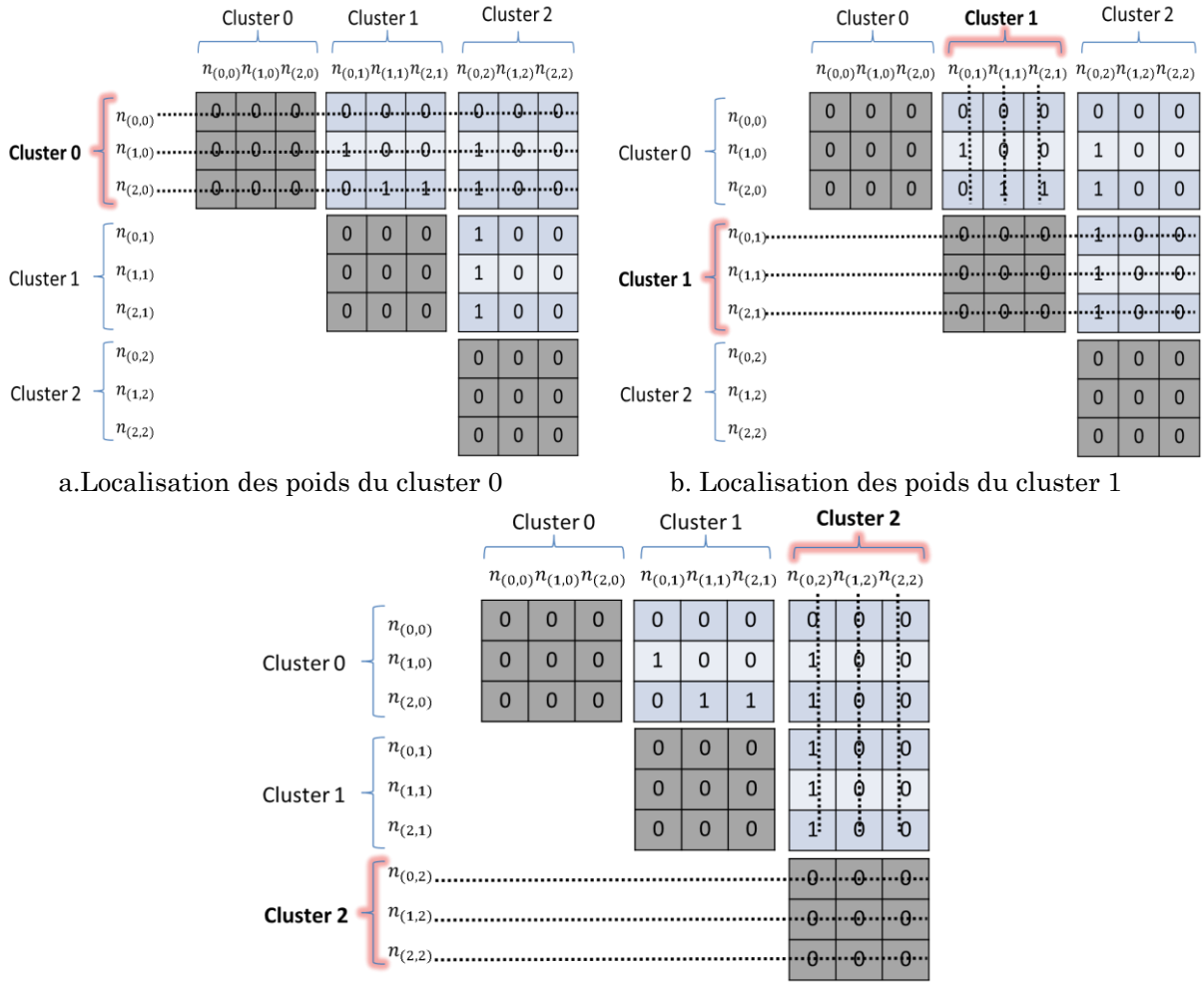
$$
\begin{aligned}
v(n_{(2,0)})(1) = &\Big(\big(v(n_{(0,1)})(0) \wedge w(n_{(2,0)}; n_{(0,1)})\big) \vee (v(n_{(1,1)})(0) \wedge w(n_{(2,0)}; n_{(0,1)}))\\
&\vee \big(v(n_{(2,1)})(0) \wedge w(n_{(2,0)}; n_{(0,1)})\big) \vee 0\Big)\\
&\wedge \Big(\big(v(n_{(0,2)})(0) \wedge w(n_{(2,0)}; n_{(0,2)})\big) \vee (v(n_{(1,2)})(0) \wedge w(n_{(2,0)}; n_{(0,2)}))\\
&\vee \big(v(n_{(2,2)})(0) \wedge w(n_{(2,0)}; n_{(0,2)})\big) \vee 0\Big)\\
v(n_{(2,0)})(1) = &\Big(\big((0 \wedge 0) \vee (1 \wedge 1) \vee (0 \wedge 1)\big) \vee 0\Big) \wedge \Big(\big((1 \wedge 1) \vee (0 \wedge 0) \vee (0 \wedge 0)\big) \vee 0\Big)\\
= &\ 1
\end{aligned}
$$

A la fin de la première itération, les valeurs des neurones $n_{(0,0)}$ et $n_{(1,0)}$ ont été évaluées comme étant 0 tandis que la valeur du neurone $n_{(2,0)}$ a été évaluée comme étant 1. Ce dernier est alors sélectionné et le réseau fourni le message originel [2,1,0].

C. Mémoire réduite

Afin de déterminer si un neurone donné est actif durant la phase de décodage, il est nécessaire de vérifier s'il est connecté avec des neurones actifs dans des clusters distants. Pour atteindre cet objectif, il a été proposé dans les modèles GBNN originels [62], [78] que chaque cluster apprenne et stocke pour chacun de ses neurones les poids synaptiques (c.à.d. w dans les équations (23) et (54)) correspondants aux connexions que ces neurones possèdent avec les neurones appartenant aux autres clusters. Ainsi, pour apprendre qu'une connexion donnée existe entre deux neurones $n_{(i,j)}$ et $n_{(i',j')}$ appartenant à des clusters différents ($j \neq j'$), deux poids doivent être stockés : premièrement dans le cluster j, où $n_{(i,j)}$ enregistre qu'il est connecté à $n_{(i',j')}$ $\left(w\left(n_{(i,j)}; n_{(i',j')}\right) = 1\right)$ et ensuite dans le cluster j', où $n_{(i',j')}$ enregistre qu'il est connecté à $n_{(i,j)}$ $\left(w\left(n_{(i',j')}; n_{(i,j)}\right) = 1\right)$. Cependant, ces deux poids sont toujours identiques car la connexion entre ces deux neurones est symétrique. Cette propriété peut être observée dans la Figure V.1 où la matrice contenant les poids synaptiques est symétrique. Ainsi, il n'est pas nécessaire de stocker les deux valeurs.

L'optimisation suppose que les matrices synaptiques sont partagées entre les clusters. Ainsi, stocker uniquement un poids synaptique par paire de neurones permet de diviser le coût de mémorisation du réseau par deux sans aucune perte de performance. De plus, afin de stocker des valeurs synaptiques dans la mémoire, des ressources logiques pour l'apprentissage sont nécessaires (décodeurs, accumulateurs, multiplexeurs, …). Mémoriser la moitié de l'information permet de se débarrasser de ces ressources de calcul et de réduire encore plus les ressources requises par l'architecture. La matrice synaptique optimisée est nommée *matrice triangulaire* dans le reste du chapitre. La Figure V.1 illustre pour chaque cluster de notre exemple les localisations des poids une fois que la matrice triangulaire est utilisée.



a.Localisation des poids du cluster 0

b. Localisation des poids du cluster 1

c. Localisation des poids du cluster 2

Figure V.1 Localisation des poids synaptiques dans la matrice

D. Communication sérialisée

Une implémentation totalement parallèle du modèle [62] telle que proposée par Jarollahi et al. [64] requiert une très grande quantité de fils pour connecter tous les neurones à une grande quantité de logique de calcul afin de calculer les valeurs de tous les neurones au même moment. En effet, dans le GBNN, un neurone $n_{(i,j)}$ d'un cluster j doit être connecté à tous les neurones de tous les clusters distants. Ainsi, chaque neurone est connecté à $L * (C - 1)$ neurones. Etant donné que chaque cluster contient L neurones, un cluster est connecté à $L * L * (C - 1)$ neurones. Finalement, si tous les clusters (c.à.d. les C clusters) sont considérés, le nombre de total de données échangées et calculées en parallèle est de $C * L * L * (C - 1) = C * L^2 * (C - 1)$.

Afin de réduire la complexité du réseau d'interconnexion et des ressources de calcul, les communications et les calculs doivent être sérialisés (c.à.d. qu'à chaque cycle, le volume d'information échangé en parallèle doit être réduit) et les calculs doivent être itératifs (c.à.d. que le calcul est étalé dans le temps et les ressources de calcul sont réutilisées). La

sérialisation permet de réduire considérablement la surface et donc de concevoir des architectures manipulant des GBNN ayant des tailles plus grandes. Cependant, celle-ci conduit aussi à l'augmentation de la *latence* de chaque itération (nombre de cycles requis pour réaliser une itération de décodage). Malgré cette augmentation, si la sérialisation est conçue de manière intelligente, celle-ci peut conduire à des fréquences plus élevées qui finalement, vont limiter son impact sur les performances en termes de temps de calcul.

La sérialisation peut être focalisée soit sur les clusters, soit sur les neurones.

Dans un mode focalisé sur les clusters (*cluster-série*), ceux-ci prennent chacun leur tour afin de diffuser la valeur de tous leurs neurones.

Dans un mode focalisé sur les neurones (*neurone-série*), les clusters diffusent en parallèle la valeur de l'un de leur neurone (c.à.d. les valeurs des neurones appartenant au même cluster sont diffusées successivement à tous les autres clusters).

Dans tous les cas, un cluster peut calculer partiellement les valeurs de ses neurones tout en recevant les valeurs courantes de ses neurones distants. Les valeurs de ses neurones locaux sont mises à jour en utilisant séquentiellement les valeurs des neurones provenant des clusters distants. Les calculs se terminent lorsque toutes les valeurs des neurones distants ont été reçues et que les valeurs locales ont été calculées.

Dans le cas d'un mode focalisé sur les clusters, une itération prendra C cycles pour s'achever tandis qu'elle prendra L cycles dans le mode basé sur les neurones. La sérialisation permet de réduire le nombre d'interconnexions car moins de données sont diffusées à chaque cycle. Cela permet aussi de réduire le coût des ressources de calcul car les éléments de calcul sont réutilisés à travers les cycles. Cependant, une implémentation directe requiert de la logique de contrôle et de routage conduisant à un surplus inacceptable en termes de coût. Ce surcout est dû à la nécessité d'utiliser des multiplexeurs pour accéder successivement aux poids synaptiques qui sont stockés dans des éléments de stockage indépendants.

La Figure V.2 montre une architecture où quatre buffers stockant les poids synaptiques sont utilisés et où le poids synaptique requis est sélectionné grâce à un ensemble de multiplexeurs durant le processus de récupération. Durant la phase d'apprentissage (resp. restitution), les données (c.à.d. les poids synaptiques) à l'entrée (resp. à la sortie) sont stockées séquentiellement dans (resp. sélectionnées des) les buffers à qui l'ordre est donné par exemple par un compteur (le compteur fonctionne selon le signal *Valide*). L'inconvénient de cette conception provient des multiplexeurs et de la logique de contrôle associée. L'ensemble conduit à une augmentation inacceptable de la surface.

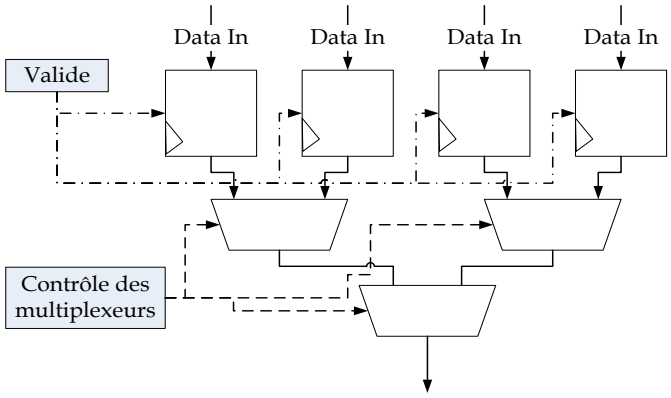


Figure V.2 Surcoût de la logique de routage dû à l'usage de multiplexeurs

Afin d'éviter cette pénalité, nous proposons l'utilisation d'un mécanisme appelé *anneau de flip-flops*. L'anneau de flip-flops est un registre à décalage circulaire (Voir Figure V.3). La taille de cet anneau est égale au nombre de cycles requis pour transférer toutes les informations (c.à.d. C ou L) en fonction du mode de sérialisation choisi (c.à.d. cluster ou neurone). Ainsi, les groupes de poids synaptiques ne sont plus stockés dans des registres indépendants mais plutôt dans des registres interconnectés. La sortie de l'anneau est connectée à la logique de calcul supprimant ainsi toute la circuiterie additionnelle.

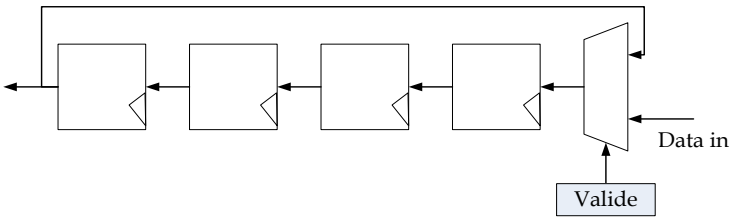
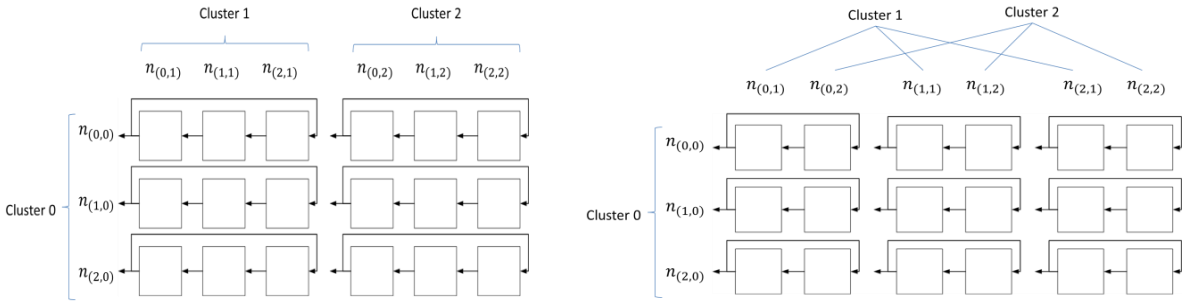


Figure V.3 Anneau de flip-flops

Dans la Figure V.3, quatre buffers sont utilisés pour stocker des poids synaptiques. Cependant, seulement un multiplexeur *2:1* est requis au lieu des *L-1* multiplexeurs *2:1* de l'architecture précédente (Voir Figure V.2). Ainsi, durant la phase d'apprentissage (phase durant laquelle les cliques et donc les poids synaptiques sont stockés dans la mémoire), la donnée d'entrée est stockée dans le premier buffer et les autres données sont déplacées dans l'anneau. Durant la phase de décodage, les données sont déplacées et les données de sortie sont toujours lues dans le dernier buffer. Après *L=4* cycles, l'anneau revient à son état initial.

La Figure V.4 (a et b) illustre comment les anneaux de flip-flops sont utilisés respectivement pour une sérialisation focalisée sur les neurones ou pour une sérialisation focalisée sur les clusters en considérant la matrice synaptique carrée présentée précédemment dans la Figure II.11. Dans la Figure V.4.a) où sont montrées uniquement les connexions entre les clusters $\mathcal{C}_0$ et ses clusters distants, il peut être observé qu'après *i* cycles du processus de décodage, que

tous les clusters (voir les lignes) ont en accès à tous les poids synaptiques que les neurones locaux partagent avec le $i^{ème}$ neurone des clusters distants. Dans la Figure V.4.b) où sont montrées uniquement les connexions entre le cluster $\mathcal{C}_0$ et ses clusters distants, il peut être observé qu'après i cycles du processus de décodage, tous les clusters auront en accès à tous les poids synaptiques que leurs neurones partagent avec tous les neurones du $i^{ème}$ cluster distant.



a) Focalisation sur les neurones b) Focalisation sur les clusters

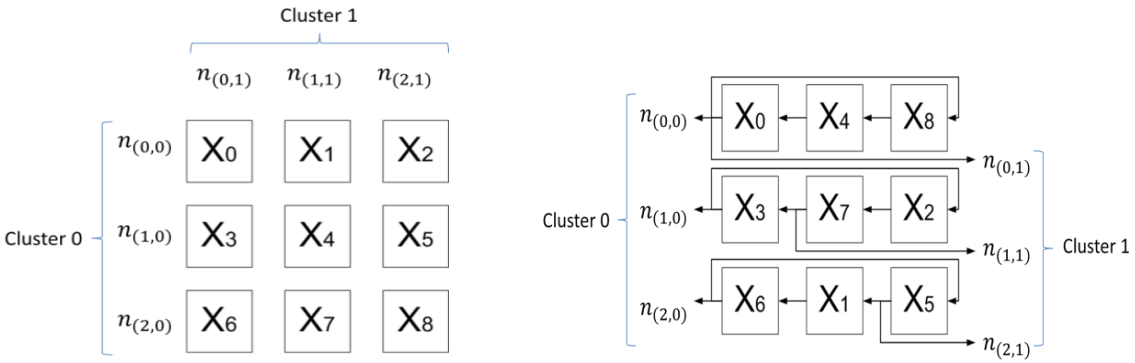
Figure V.4 Anneau de flip-flops appliqué à la matrice *carrée* et selon différents modes de sérialisation

Les anneaux de flip-flops peuvent aussi être combinés avec les matrices triangulaires. Cependant, les données doivent être placées de manière à éviter toute logique additionnelle pénalisante. En effet, la matrice triangulaire est partagée entre les clusters et ceux-ci ont accès deux à la fois aux même poids synaptiques. Dans une sérialisation focalisée sur les neurones, ce comportement peut être observé dans la Figure V.1.a) qui présente uniquement les poids partagés par les clusters $\mathcal{C}_0$ et $\mathcal{C}_1$. Si l'on considère le cycle auquel tous les clusters diffusent la valeur de leur neurone n_0 (c.à.d. $n_{(0,0)}$, $n_{(0,1)}$ et $n_{(0,2)}$), alors le cluster $\mathcal{C}_0$ accède à tous les poids qui sont liés aux neurones distants n_0, ce qui signifie la première ligne de la Figure V.4. Déplacer les données horizontalement serait correct pour $\mathcal{C}_0$ mais pas pour $\mathcal{C}_1$ tandis que déplacer les données verticalement serait correct pour $\mathcal{C}_1$ mais pas pour $\mathcal{C}_0$.

La solution consiste à positionner les données dans les anneaux de flip-flops de telle sorte qu'à chaque cycle, chaque cluster d'une paire donnée puisse accéder à la bonne information au bon moment. De cette manière, tous les clusters pourront accéder à leurs données de manière parallèle en lisant toujours le même endroit afin d'éviter l'ajout d'une logique additionnelle coûteuse. Afin d'atteindre cet objectif, considérons deux matrices $L * L$ appelées *POIDS* et *ANNEAU*, *POIDS* représentant la matrice de poids synaptiques partagée par deux clusters et *ANNEAU* représentant l'ensemble des L anneaux de flip-flops de L cellules chacune. Considérons deux indices $M, N \in [0; L-1]$. Alors en appliquant (55), il est possible de calculer la position où les données doivent être introduites dans les cellules de l'anneau de flip-flops.

$$ANNEAU(M,N) = POIDS(M + N \ mod \ L, N) \tag{55}$$

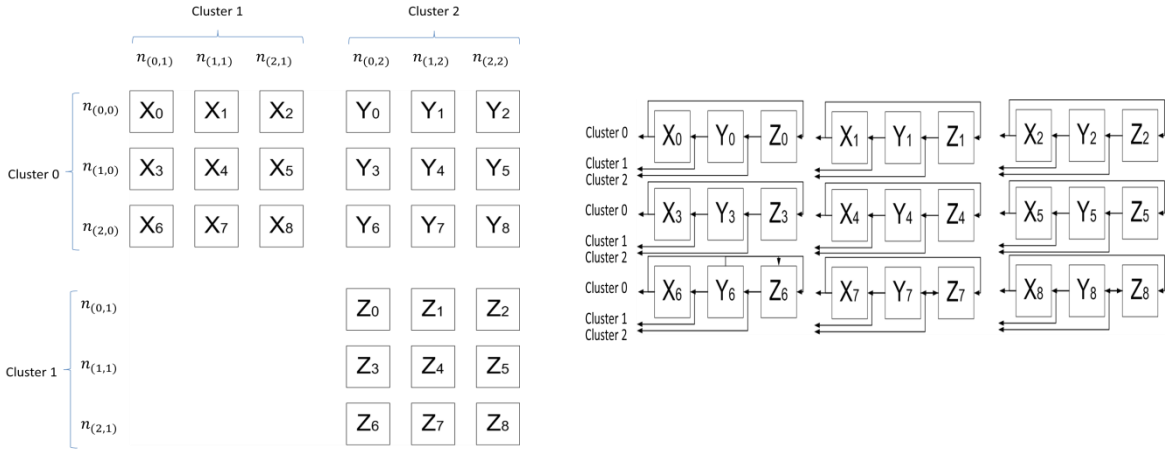
Cette équation permet le déplacement par N lignes des données de la $N^{ième}$ colonne de la matrice *POIDS* afin de les positionner dans la matrice *ANNEAU*. Pour des raisons de simplicité, la Figure V.5.a) illustre le positionnement obtenu en appliquant (55) afin de combiner les anneaux de flip-flops et la sérialisation focalisée sur les neurones. Par exemple, $ANNEAU(0,2) = POIDS(2,2) = X_8$. On peut observer qu'après i cycles du processus de décodage, le cluster $\mathcal{C}_0$ (resp. $\mathcal{C}_1$) aura accès aux poids synaptiques que ses neurones locaux partagent avec au $i^{ème}$ neurone du cluster $\mathcal{C}_1$ (resp. $\mathcal{C}_0$).



a) Focalisation sur les neurones b) Focalisation sur les clusters

Figure V.5 Sérialisation focalisée sur les neurones et sur la matrice triangulaire

Si l'on considère la sérialisation focalisée sur les clusters, les anneaux de flip-flops doivent récupérer les poids synaptiques des différentes paires de clusters. La Figure V.6.a) et la Figure V.6.b) illustrent des matrices triangulaires avec des poids qui sont identifiés par des lettres indexées X, Y et Z. La Figure V.6.b) montre l'ensemble des anneaux de flip-flops qui permet après i cycles du processus de décodage, que chaque cluster accède à tous les poids synaptiques qu'il partage avec les neurones du $i^{ème}$ cluster distant.



a) Poids synaptiques b) Anneau de flip-flops

Figure V.6 Sérialisation focalisée sur les clusters

E. Expériences

L'intérêt des optimisations proposées est montré à travers plusieurs expériences où le modèle et l'architecture originelle sont comparés au modèle et aux architectures proposées. Trois expériences ont été réalisées :

- Un premier ensemble d'études théoriques a pour objectif de montrer que notre modèle totalement binaire améliore la performance du GBNN originel [62] et atteint celle du modèle amélioré proposé dans [101].
- Le second ensemble d'expériences se focalise sur la conception matérielle. L'état de l'art et les architectures optimisées sont comparées. Basée sur une librairie technologie STM Microelectronics 90nm, de nombreux réseaux sont explorés et leurs complexités architecturales sont analysées.
- Le troisième et dernier ensemble d'expériences compare les résultats de synthèse basés sur un FPGA Stratix IV et des plateformes HardCopy de Altera pour différentes architectures de réseaux.

Afin de permettre une lecture facile des courbes, le concept de *versioning* a été usé afin de nommer les architectures.

- Les architectures basées sur le modèle de GBNN originel [64] sont référencées comme étant *V0 ;*
- Les architectures basées sur le modèle totalement binaire que nous avons proposées sont référencées comme étant *V1.0 ;*
- Les architectures basées sur le modèle binaire et sur la matrice de poids synaptiques triangulaire sont référencées comme étant *V1.1 ;*
- Les architectures basées sur le modèle binaire et sur la sérialisation focalisée sur les clusters sont référencées comme étant *V1.2 ;*
- Les architectures basées sur le modèle binaire et sur la sérialisation focalisée sur les neurones sont référencées comme étant *V1.3.*

1. Analyse de la performance

Afin d'étudier et de comparer les performances de décodage de notre modèle totalement binaire, des simulations Matlab ont été réalisées. Ces simulations ont permis de déduire le TED en fonction du nombre de messages appris quand 50% de clusters ne contiennent pas d'information. Ces expériences ont été réalisées pour un réseau contenant C=8 clusters et L=256 neurones (c.à.d. un total de 1024 neurones). Quatre itérations de décodage ont été utilisées car il a été montré dans [62] que ce nombre permet à un GBNN d'obtenir sa meilleure performance. Les simulations ont été réalisées sur un core i7 M620 2.7Ghz avec 4Go de RAM. Les résultats sont comparés avec le GBNN originel de [62] et celui de [101].

La Figure V.7 montre le TED en fonction du nombre de messages appris. Il peut être observé que le modèle GBNN défini dans [101] a une meilleure performance que celui du modèle GBNN originel [62]. En effet, jusqu'à 20.000 messages, les meilleurs réseaux sont capables d'atteindre au maximum 10% d'erreur de décodage tandis que la performance du modèle originel commence à se détériorer à partir de 15.000 messages. De plus, notre contribution possède la même performance que la méthode de décodage proposé dans [101] car les courbes se superposent parfaitement. Cela confirme que notre modèle totalement binaire ne dégrade pas la bonne performance des modèles GBNN à décodage entier.

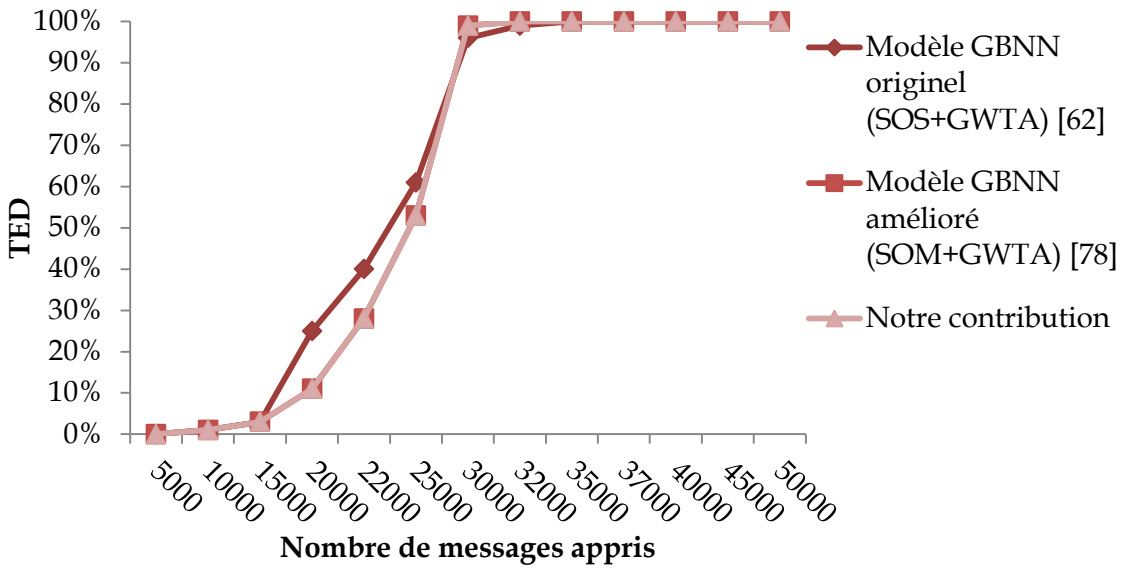


Figure V.7 Taux d'erreur de décodage pour différents modèles

2. Analyse de la complexité

Le second ensemble d'expériences explore un vaste ensemble d'architectures afin d'étudier la réduction en complexité offerte par les optimisations que nous proposons. Les résultats sont donnés en termes de *Portes NAND Equivalentes* (PNE) basées sur la librairie 90nm des cellules de St Microelectronics. L'architecture V0 est utilisée comme base de comparaison.

La Figure V.8 montre un premier ensemble de résultats dans lequel le nombre de clusters et le nombre de neurones par cluster varient (Variation de C et L de 1 à 128). Il peut être noté que la Figure V.8 inclut une borne supérieure (c.à.d. $50*L^2C^2$) afin d'observer les limites de l'espace de conception plus facilement. L'analyse des résultats montre que toutes les architectures sont nettement plus petites que les architectures V0 de référence. En effet, alors que l'usage du modèle binaire réduit la surface totale de V1.0 de moitié, les combinaisons de toutes nos améliorations proposées poussent l'optimisation plus loin. V1.1 réduit la complexité de l'architecture de 1/3 de V0 (c.à.d. 70% de réduction de surface), tandis que V.1.2 et V1.3 la réduisent à 1/6 (c.à.d. 83% de réduction de surface). Le coût total en ressources présenté en Figure V.8 se décompose chacun en trois parties : (1) les ressources de

mémorisation qui sont représentées par les poids synaptiques, (2) les ressources d'apprentissage qui sont représentées par les ressources nécessaires pour réaliser l'apprentissage d'un message et (3) les ressources de calcul qui sont représentées par les ressources nécessaires pour réaliser le décodage.

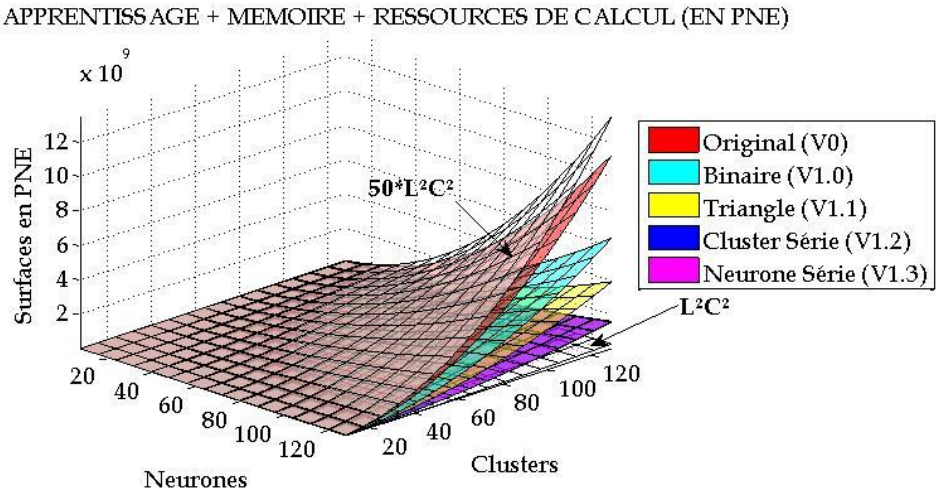
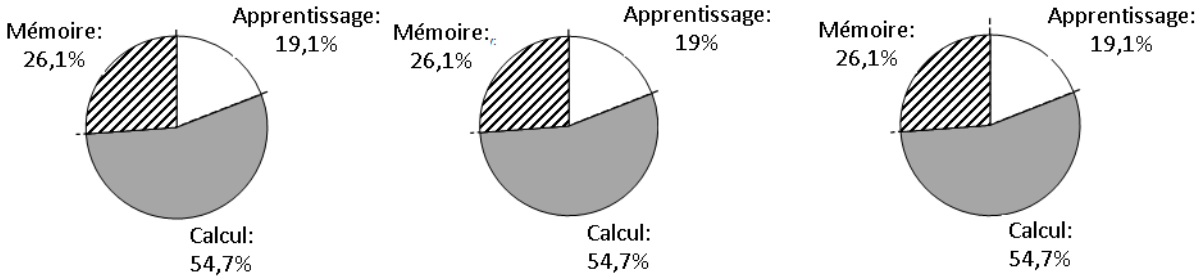


Figure V.8 Surface pour les réseaux jusqu'à (C,L)=(128,128) (en portes NAND équivalentes)

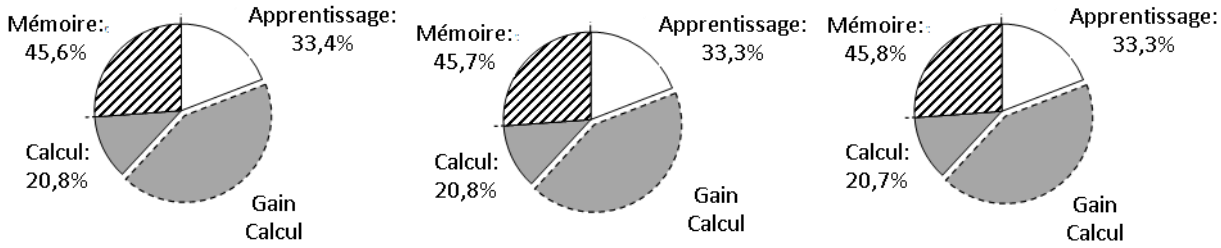
Les Figure V.9, Figure V.10, Figure V.11 montrent la répartition de ces trois contributeurs pour trois configurations de réseaux : 256 clusters/64 neurones, 128 clusters/128 neurones et 64 clusters/256 neurones. Les ressources de mémorisation sont présentées en gris sombre, les ressources d'apprentissage en blanc et les ressources de calcul en stries. Ainsi, la Figure V.9 décrit la répartition des ressources dans V0 et montre que la surface dépend surtout des ressources de calcul (54,7% du coût total en moyenne).



a) C= 256 L=64 b) C= 128 L=128 c) C= 64 L=256

Figure V.9 Répartition des ressources dans les architectures V0

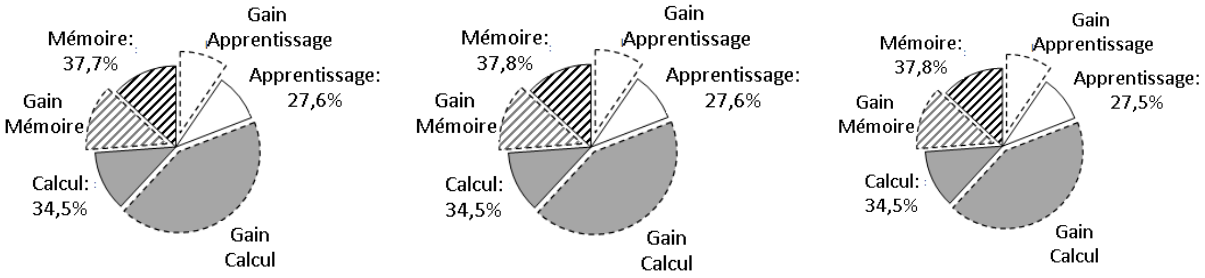
La Figure V.10 montre la répartition des ressources dans V1.0 et mentionne à travers des partitions avec des lignes en traits interrompus les gains obtenu par rapport aux résultats de l'architecture de référence. Il peut être observé que le modèle totalement binaire réduit énormément le coût des ressources de calcul. Ces ressources représentent presque 20% de la surface totale dans V1.0 (contre 54,7% en moyenne dans V0). Presque la moitié de l'architecture est maintenant composée des ressources de mémorisation.



a. C= 256 L=64 b. C= 128 L=128 c. C= 64 L=256

Figure V.10 Répartition des ressources dans les architectures V1.0

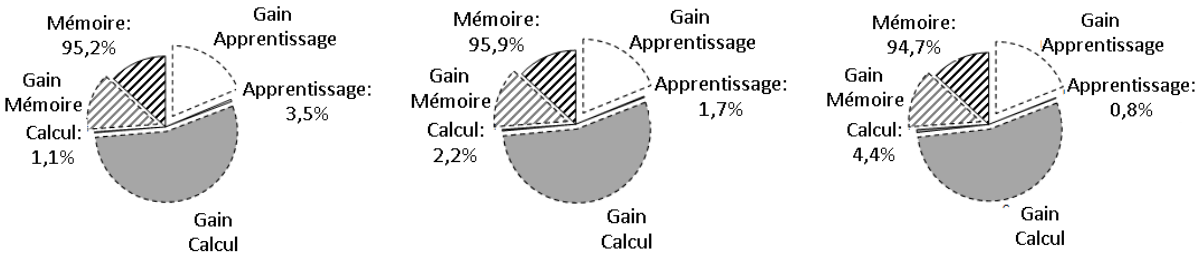
La Figure V.11 illustre la répartition des ressources dans l'architecture V.1.1. Coupler la matrice de poids synaptiques triangulaire au modèle totalement binaire permet de réduire encore plus la surface totale. En plus des économies obtenues en utilisant le modèle totalement binaire (c.à.d. V1.0), les coûts des ressources d'apprentissage et de mémoire ont été respectivement réduits de 10% et presque 50%. Les ressources d'apprentissage représentent maintenant près de 27% de la surface totale au lieu de 33,4% dans V1.0 alors que les ressources mémoire représentent 37,8% en moyenne au lieu 45,7% dans V1.0.



a. C= 256 L=64 b. C= 128 L=128 c. C= 64 L=256

Figure V.11 Répartition des ressources dans les architecturesV1.1

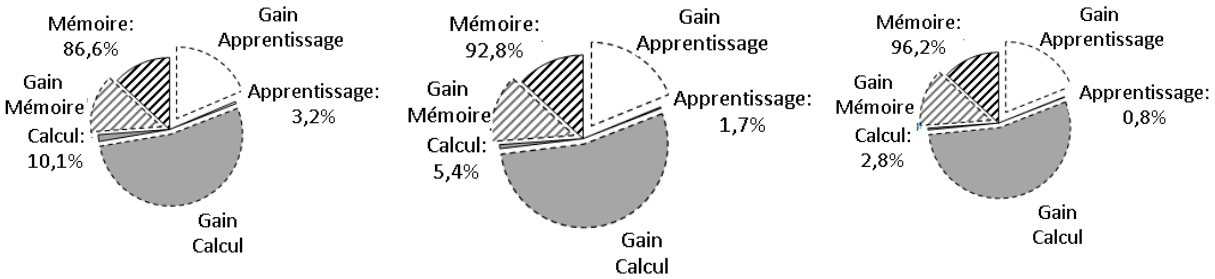
La Figure V.12 et la Figure V.13 représentent respectivement la répartition des ressources dans les architectures V1.2 et dans V1.3. Les résultats montrent une importante réduction par rapport à V1.0 et à V1.1 et encore plus par rapport à V0. Ceci est une démonstration que la sérialisation est capable non seulement de réduire la complexité des réseaux d'interconnexion, mais aussi d'avoir un grand impact sur la surface totale avec une source limitée de ressources de calcul. De plus, avec ces modèles et optimisations proposés, les ressources de mémorisation représentent près de 95% de la surface totale. Cela signifie que le réseau de neurones est maintenant principalement composé d'éléments de mémorisation.



a. C= 256 L=64 b. C= 128 L=128 c. C= 64 L=256

Figure V.12 Répartition des ressources dans les architectures V1.2

Il peut aussi être observé que si le nombre de clusters est plus grand que le nombre de neurones, la proportion des ressources de calcul dans les architectures V1.2 (binaire cluster-série) est plus faible que dans les architectures V1.3 (binaire neurone-série). Dans le cas contraire, si le nombre de clusters est plus faible que le nombre de neurones, la proportion des ressources de calcul dans les architectures V1.2 est plus faible que dans les architectures V1.3.



a. C= 256 L=64 b. C= 128 L=128 c. C= 64 L=256

Figure V.13 Répartition des ressources dans les architectures V1.3

3. Résultats de synthèse

a) Cible FPGA

Après avoir évalué la surface, nous avons conçu et synthétisé sur une plateforme Stratix IV FPGA, un ensemble d'architectures. Afin de permettre des comparaisons justes, toutes les architectures ont été réalisées sur la même cible que celle utilisée dans [64] c.à.d. le FPGA Stratix EP4SGX230KF40C2. Différentes tailles de réseaux ont été explorées allant de C*L=2*2 à C*L=16*16. Le cas C*L=8*16 est la configuration utilisée dans [64]. Nous avons fait augmenté la complexité jusqu'à 256 neurones parce que l'architecture de référence V0 atteint la limite du FPGA pour cette taille.

La Figure V.14 montre l'évolution de l'usage du nombre de registres en fonction du nombre de neurones et du nombre de clusters dans le réseau. On peut observer que le modèle totalement binaire V1.0 requiert moins de ressources que l'architecture de référence V0 alors que la matrice de poids synaptiques de V1.1 permet d'atteindre une réduction de 50% par rapport à toutes les autres configurations.

La Figure V.15 présente les résultats de la synthèse en termes de LUTs et démontre l'intérêt des optimisations proposées. Par comparaison avec l'architecture de référence V0, la réduction en surface varie de 62% pour V1.0 jusqu'à 86% pour V1.2. Plus la taille du réseau augmente, plus la réduction est importante. En considérant une configuration 16*16, l'économie en surface atteint respectivement 76% et 92% pour les architectures V1.0 et V1.2.

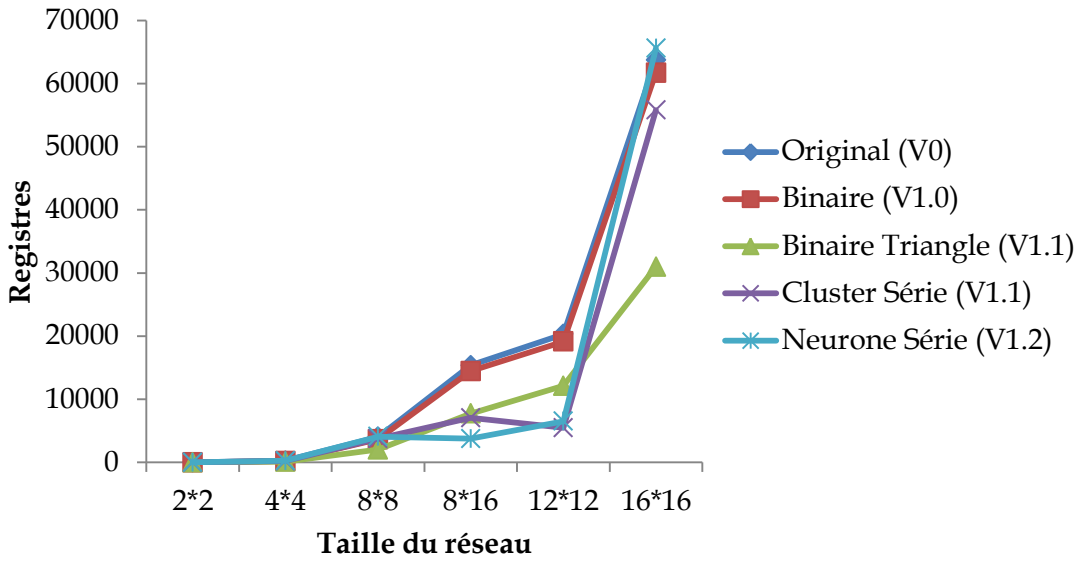


Figure V.14 Usage des registres du FPGA

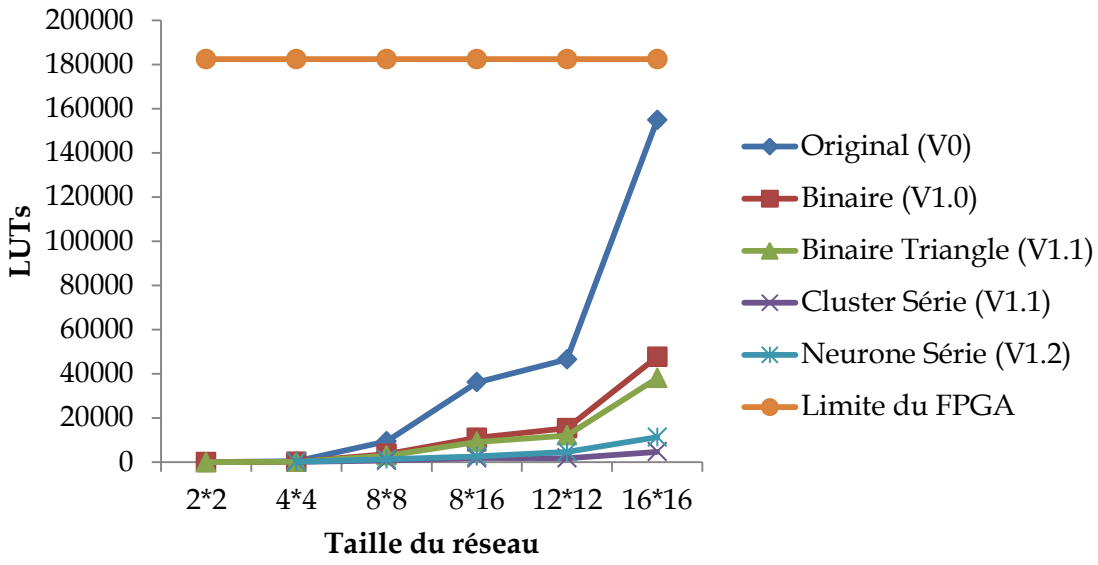


Figure V.15 Utilisation des LUTs du FPGA

La Figure V.16 résume ces résultats en présentant le pourcentage d'utilisation des ressources du FPGA et confirme la tendance générale : plus le réseau grandit, plus l'écart entre les architectures proposées et l'architecture de référence augmente. Pendant que V0 atteint

rapidement 100% de l'usage de la FPGA, les architectures les plus optimales V1.2 et V1.3 utilisent moins de 8% des ressources.

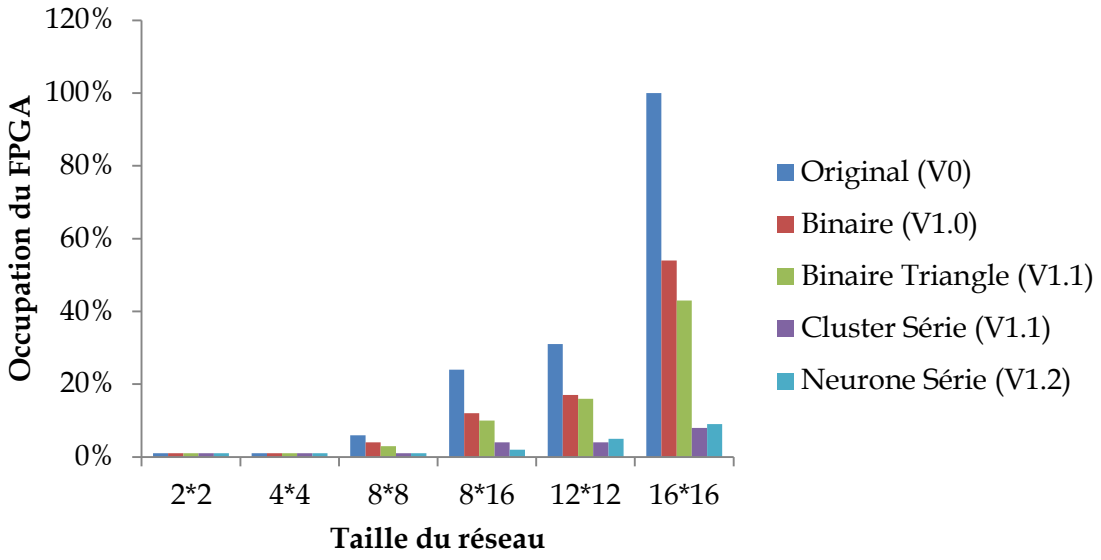


Figure V.16 Taux d'utilisation des ressources du FPGA

La Figure V.17 montre l'évolution des fréquences d'horloge en fonction de la taille des réseaux. On peut observer que les fréquences d'horloge décroissent plutôt rapidement pour les petits réseaux alors que les performances décroissent plus lentement pour les réseaux les plus larges. Les architectures de référence V0 décroissent plus rapidement que les autres architectures. En considérant le réseau le plus large, V1.3 obtient la meilleure performance par rapport à toutes les autres architectures avec une fréquence d'horloge de 338,64MHz. La bonne performance en timing des architectures sérialisées permet d'équilibrer leur comportement séquentiel.

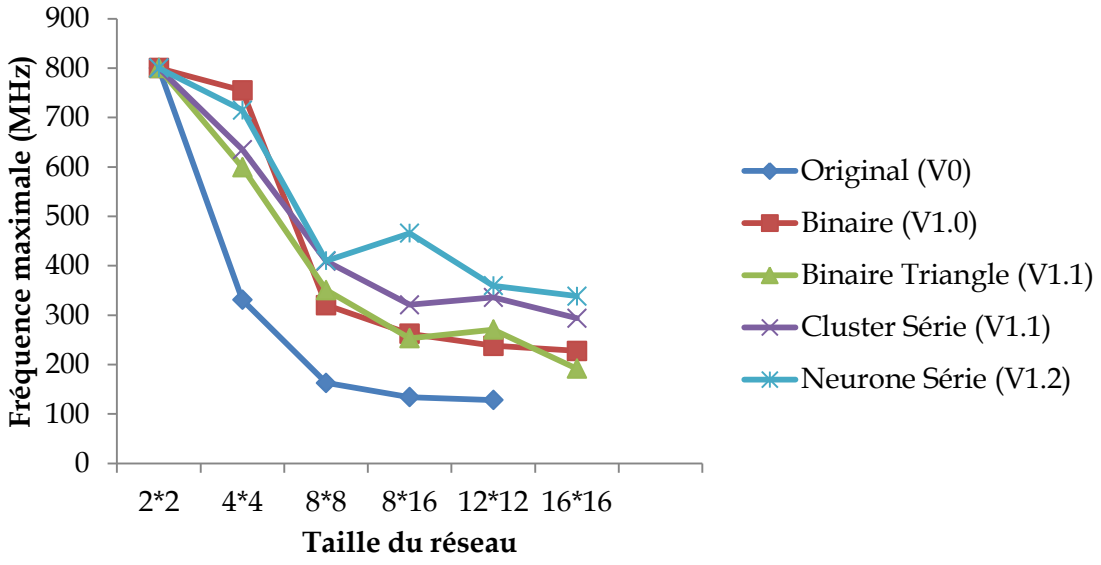


Figure V.17 Fréquences d'horloge des différentes architectures

b) Cible ASIC

La technologie ASIC a été ciblée en utilisant la plateforme HardCopy qui mappe toutes les ressources utilisées dans l'architecture en HCELLs au lieu d'utiliser des LUTs et des registres.

La Figure V.18 présente les résultats de synthèse en termes de HCELLs. Comparé aux architectures de référence V0, les réductions en surface varient de 33% pour V1.0 jusqu'à 57% pour V1.2 lorsque la configuration 16*16 est considérée.

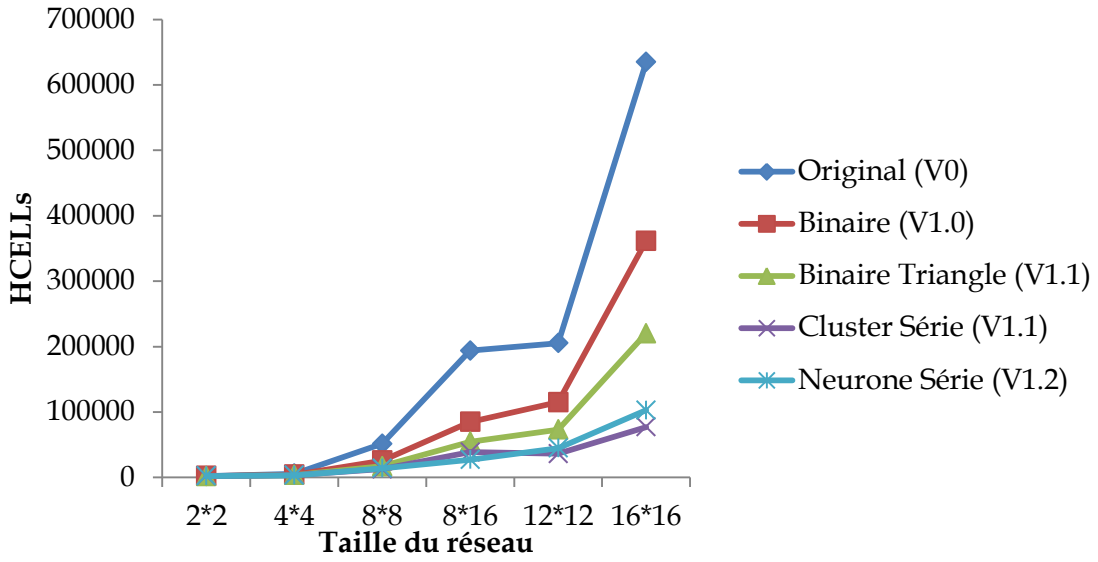


Figure V.18 Résultats en termes de surface (HCELLs)

La Figure V.19 présente l'évolution des fréquences maximales. La fréquence maximale de V0 décroit plus rapidement que les autres tandis que celles-ci ont de meilleures fréquences. A part les configurations 8*8 et 16*16, les architectures V1.3, V1.0 et V1.1 obtiennent de temps en temps des performances équivalentes.

Tous les résultats de synthèse vérifient les tendances démontrées par l'analyse de la complexité et confirme les bons résultats que nos optimisations permettent d'atteindre.

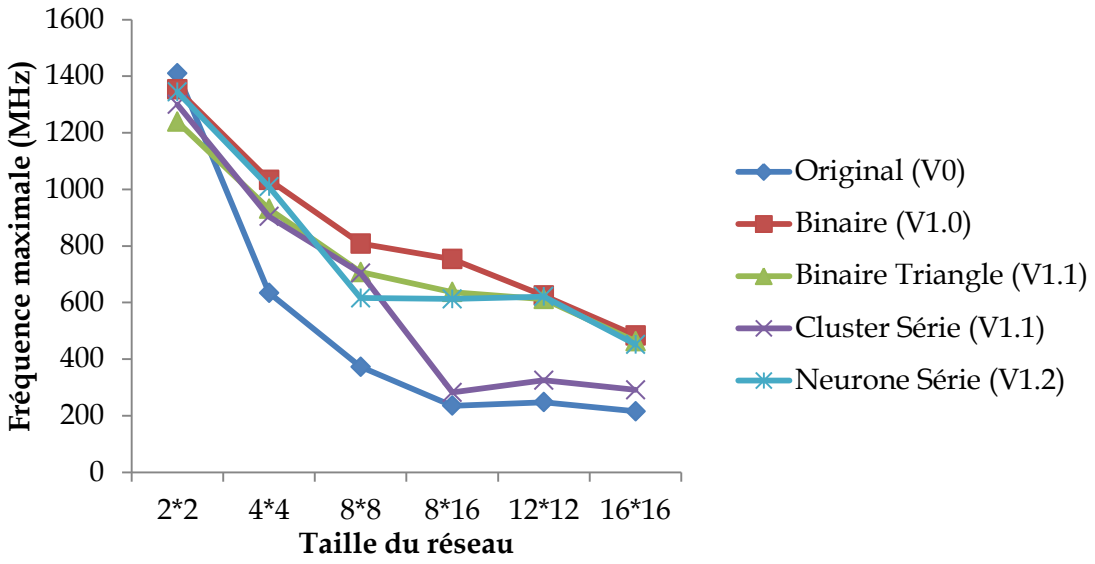


Figure V.19 Fréquences d'horloge des différentes architectures (ASIC)

F. Bilan

Dans cette partie, des optimisations pour réduire la complexité des architectures du modèle GBNN et permettant la conception d'architectures matérielles peu coûteuses, ont été présentées.

Premièrement, un modèle totalement binaire a été défini. Il permet de simplifier le modèle originel et de réduire les ressources de calcul dans les architectures matérielles. Ensuite, le stockage de la moitié des poids synaptiques par rapport à l'architecture originelle a été proposé, réduisant ainsi le coût des éléments de stockage et la complexité du processus d'apprentissage. Finalement, des communications sérialisées ont été introduites entre les neurones afin de permettre plus d'optimisation de la complexité. La combinaison de ces trois axes d'optimisations n'est pas évidente. Les résultats des évaluations ont montré que les optimisations permettent d'obtenir des réductions de surface importantes ainsi que de meilleures fréquences d'horloge comparées aux architectures de référence sans perte de performance et de fonctionnalité par rapport au modèle GBNN originel.

En comparant l'architecture NOC proposée dans [86] pour le GBNN originel et les architectures présentées de ce chapitre, on peut constater que les deux architectures sont sérialisées en termes de calcul et de communication. D'un côté, l'architecture NOC utilise un réseau d'interconnexion moins dense que les architectures proposées (réseau en grille pour le NOC contre réseau maillé pour les architectures proposées). D'un autre côté, l'architecture NOC est plus lente à cause des temps de création et de transfert de paquets de donnée tandis que les architectures proposées dans ce chapitre transfèrent directement les informations et ont un temps de communication et de calcul constant.

Toutes ces architectures demeurent cependant des architectures spécialisées qui stockent intégralement la matrice d'adjacence, ce qui est leur principale limitation en termes de taille de réseau mis en œuvre. Dans le prochain chapitre, une architecture générique est proposée et est capable d'utiliser un mode de stockage des matrices d'adjacence moins coûteux et moins contraignant qu'un stockage intégral.

CHAPITRE VI. Architecture Générique de Réseaux à Clones

Sommaire du Chapitre :

Dans ce chapitre, un nouveau mode de stockage des matrices parcimonieuses et une architecture générique permettant de mettre en œuvre des réseaux à clones de tailles variables sont présentées. Des améliorations de cette architecture sont aussi proposées.

A. Introduction

Les architectures spécialisées mettant en œuvre le GBNN possèdent les limitations suivantes :

- Une fois synthétisées, elles ne peuvent manipuler que des réseaux de taille prédéfinie ;
- Les architectures sont coûteuses ce qui implique que la taille des réseaux mis en œuvre est faible ;
- Elles ne gèrent que des réseaux ayant une organisation particulière (auto-associatif ou hétéro-associatif) ;

Cette limitation de la taille des réseaux mis en œuvre est due à deux facteurs :

- La matrice de poids synaptiques qui :
    - Est allouée intégralement par rapport à un réseau de taille définie [64], [65], [102]. En effet, toutes les mises en œuvre utilisent un mode de représentation *bit-vector* qui ne supporte pas une allocation dynamique de la mémoire.
    - Est coûteuse avec une croissance quadratique de son coût (22) en fonction du nombre de clusters et du nombre de neurones (cas du GBNN).
    - Est souvent mise en œuvre dans des registres au lieu d'être mis en œuvre dans les mémoires RAM. En effet, ces dernières permettent de stocker plus de bits que les registres disponibles sur les FPGAs (plusieurs centaines de milliers de bits contre plusieurs Mbits). Ces RAMs sont aussi capables de communiquer avec des mémoires externes largement plus grandes (plusieurs Gbits) bien que plus lentes.
- Les ressources de calcul qui :
    - Sont définies par rapport à un réseau de taille prédéfinie ;
    - Ont un coût qui augmente en fonction de la taille du réseau mis en œuvre.

Il est alors nécessaire de proposer une architecture générique où :

- La matrice de poids synaptiques utilise un mode de représentation permettant d'une part l'usage de l'allocation dynamique de la mémoire et d'autre part utilisant le moins de mémoire possible sans augmenter la complexité à d'accès à la matrice ;
- Les ressources de calcul sont définies à la conception et sont compatibles pour différentes tailles de réseaux. C.à.d. qu'à la conception, un certain nombre d'éléments de calcul est présent et permet de manipuler des réseaux de différentes tailles.
- Des réseaux à clones ayant différentes tailles et différentes organisations (auto-associatif, hétéro-associatif, …) peuvent être mis en œuvre ;

Dans ce chapitre, un mode de stockage moins couteux en mémoire que les modes classiques et une architecture générique compatible avec ce mode de stockage et intégrant toutes ces contraintes sont présentés.

Dans l'architecture, la règle de calcul de la valeur de chaque clone est le Sum of Max (SOM) (29) associé à un WTA. Cette règle est choisie car elle permet d'éliminer des bruits de type effacement ainsi que des bruits de type inversion.

Les différentes variantes de réseaux à clones proposés (Voir CHAPITRE III) ont une organisation auto-associative. Cependant, comme le GBNN [103], elles peuvent être étendues pour avoir une organisation hétéro-associative ou bidirectionnelle. Ainsi, après avoir présenté l'architecture générique permettant de mettre en œuvre des réseaux à clones, des pistes sont données afin que la même architecture puisse manipuler des réseaux ayant une organisation hétéro-associative ou bidirectionnelle.

Pour cette architecture, l'apprentissage est hors ligne, c.à.d. que la matrice d'adjacence du réseau est fournie. Ainsi, la présentation de l'architecture se focalise uniquement sur une architecture réalisant le décodage, ce qui est une limitation.

B. Mode de représentation optimisé des matrices parcimonieuses

Dans cette section, un nouveau mode de représentation des matrices parcimonieuses est présenté et améliore les performances des modes classiques en termes de *coût de mémorisation* (nombre total de bits requis pour stocker la matrice). Le gain en coût de mémorisation est supérieur au surplus en *complexité temporelle* par rapport aux modes classiques supportant l'allocation dynamique (CSR, CSC). Ce mode s'inspire du mode CSC (Voir CHAPITRE IV.D.4) afin d'être compatible avec l'architecture de Dorrance et al [92] (Voir CHAPITRE IV.E) nous servant de base pour concevoir l'architecture générique.

1. CSC par Groupe (CSC-G)

a) Définition

Une matrice représentée à l'aide d'une liste de lignes pour chaque colonne CSC-G utilise les trois vecteurs suivants : *pointeurs*, *lignes* et *données*.

Dans une représentation CSC-G, chaque colonne de la matrice est divisée en blocs de plusieurs lignes. Chaque bloc contient le même nombre de lignes.

Sachant que chaque élément du vecteur *lignes* doit pouvoir indiquer la position d'une valeur se trouvant sur la colonne, alors celui-ci utilise un codage sur $Nbv(u)$ bits. Le principe du

CSC-G est de représenter chaque position sur la colonne par rapport au groupe auquel il appartient (adressage relatif).

Si l'on dispose de b blocs, et sachant que le nombre de lignes de la matrice est u, alors $u = b * w$. Ainsi, chaque élément de *lignes* peut être représenté sur $Nbv(w)$ bits au lieu de $Nbv(u)$ bits comme dans le mode CSC avec $Nbv(w) < Nbv(u)$.

Afin de pouvoir séparer les éléments de chaque bloc, la méthode la plus simple est d'insérer des séparateurs entre ces blocs. Ce séparateur est un symbole spécial dont la valeur le rend discernable des autres éléments de *lignes*. Cela implique que le codage de ces éléments doit être étendu pour intégrer le codage de ce symbole. Ainsi, au lieu d'avoir un codage sur $Nbv(w)$ bits, ces éléments auront un codage sur $Nbv(w+1)$ bits.

Le parcours d'une matrice selon ce mode se déroule de la manière suivante. Pour chaque colonne j de la matrice, l'adresse de ses lignes associées $Adr(j)$ est lue dans *pointeurs*. La correspondance vers le vecteur *lignes* n'est pas directe à cause des séparateurs. Ainsi, il faut ajuster l'adresse en intégrant les séparateurs. Cela revient à calculer l'adresse $Adr'(j)$ telle que: $Adr'(j) = Adr(j) + w * j$. Ainsi, les éléments de la colonne j pourront être lus à la bonne adresse. D'un autre côté, pour chaque élément lu dans *lignes*, celui-ci doit être recodé afin de correspondre à la bonne ligne de la matrice contenant l'élément non nul (passage en adressage global).

Exemple : (*Sep* = Séparateur)

Matrice	Représentation
$\begin{bmatrix}1 & 0 & 4 & 0\\ 3 & 7 & 0 & 0\\ 0 & 0 & 2 & 9\\ 5 & 8 & 0 & 7\end{bmatrix}$	$pointeurs = [0\ 2\ 4\ 6\ 9]$ $lignes = [0\ 1\ Sep\ 1\ Sep\ 1\ Sep\ 1\ Sep\ 0\ Sep\ 0\ Sep\ \ Sep\ 0\ 1\ ]$ $données = [1\ 3\ 5\ 7\ 8\ 4\ 2\ 9\ 7]$

b) Caractérisation

Le vecteur *lignes* est augmenté de $v * (b-1)$ éléments car $b-1$ séparateurs sont utilisés pour chaque colonne de la matrice. Ainsi, la complexité du mode CSC-G est de :

$$Cx(W) = v * (b-1) + d(W) * |W| \tag{56}$$

D'un autre côté, le codage des éléments de *lignes* passe de $Nbv(u)$ à $Nbv(w+1)$ bits. Le coût total en mode CSC-G est alors de :

$$\begin{aligned} Ct(M) = {} & (v+1) * Nbv(|W|) + d(W) * |W| * Nbv(w+1) + v * (b-1) \\ & * Nbv(w+1) + d(W) * |W| * Nbc(W)\ bits \end{aligned} \tag{57}$$

Un mode *CSR-G* peut aussi être défini en se basant sur le mode *CSR*.

2. Comparaison du CSC-G aux autres modes

Le Tableau 5 résume le coût mémoire et la complexité selon les différents modes de représentation.

Tableau 5 Récapitulatif des différents modes de représentations des matrices parcimonieuses

Modes de représentation	**Coût mémoire** $Ct(W)$	**Complexité** $Cx(W)$
Bit-vector	$\|W\| + d(W) * \|W\| * Nbc(W)$	$\|W\|$
COO	$d(W) * \|W\| * Nbv(u) + d(W) * \|W\| * Nbv(v) + d(W) * \|W\| * Nbc(M)$	$d(W) * \|W\|$
CSR	$(u+1) * Nbv(\|M\|) + d(W) * \|W\| * Nbv(v) + d(W) * \|W\| * Nbc(W)$	$d(W) * \|W\|$
CSC	$(v+1) * Nbv(\|W\|) + d(W) * \|W\| * Nbv(u) + d(W) * \|W\| * Nbc(W)$	$d(W) * \|W\|$
CSC-G	$(v+1) * Nbv(\|W\|) + d(W) * \|W\| * Nbv(w+1) + v * (b-1) * Nbv(w+1) + d(W) * \|W\| * Nbc(W)$	$v * (b-1) + d(W) * \|W\|$

Ce tableau nous montre que les modes *COO*, CSC et *CSR* possèdent la plus petite complexité par rapport aux autres modes. Pour que la complexité du mode CSC-G soit proche de celle du mode *CSR* (cas optimal), il faut que $d(W) * |W| + v * (b-1)$ soit proche de $d(W) * |W|$. Or $|W| = u * v \Rightarrow d(W) * |W| + v * (b-1) = v * \big(d(W) * u + (b-1)\big)$. Cela implique que le mode CSC-G n'est intéressant en termes de complexité (mais aussi de coût car la même condition rend le coût des séparateurs négligeable par rapport au coût des données utiles) si et seulement si :

$$d(W) * u \gg (b-1) \tag{58}$$

Ainsi, à partir de la densité d'une matrice ($d(W)$) et de sa taille (u ou v), il est possible de déterminer quelle est la valeur de b permettant d'obtenir un gain en mémoire. (58) montre aussi que le mode CSC-G a un intérêt qui dépend uniquement des propriétés de la matrice en termes de taille et de densité mais pas en termes de constitution (répartition des valeurs non nulles dans la matrice).

L'intérêt pour le mode CSC-G peut être défini comme étant l'obtention d'un gain en coût de mémorisation supérieur au surplus en complexité temporelle.

Afin d'évaluer cet intérêt, le gain en coût de mémorisation et le surplus en termes de complexité temporelle ont été évalués en fonction de la densité et en fonction de la taille de la matrice (nombre de lignes u). Dans ces calculs, le coût du vecteur *données* n'a pas été pris en compte car ce vecteur est incompressible quelques soit le mode de représentation. Il est aussi difficile à prendre en compte à cause du codage des données qui représente un paramètre supplémentaire dans la comparaison. De plus, dans le cas de matrices binaires, le vecteur *données* est inexistant.

Les Figure VI.1 et Figure VI.2 montrent le résultat de ces évaluations. Les courbes en trait continu représentent le gain mémoire. Les courbes en traits interrompus représentent le surplus en complexité. Les courbes de même couleur sont liées à la même densité. Chaque courbe est liée à une densité précise. Trois densités ont été choisies : $0{,}01\%, 0{,}1\%$ et 1%. Ces densités ont été choisies dans la marge des densités des matrices évaluées dans [98] pour comparer le coût des modes de représentations des matrices parcimonieuses. Un TED faible du GBNN est obtenu pour des densités de la matrice d'adjacence inférieures à 30%. Cependant, une densité de 1% est suffisante pour obtenir des résultats intéressants.

Les résultats montrent que pour de petites matrices (< 20.000 lignes), un gain positif en coût de mémorisation n'est pas toujours garanti. Il dépend de la densité et de la taille de la matrice. Pour une densité $d(W) = 0{,}01\%$, le gain positif en mémoire n'est jamais obtenu tandis que pour $d(W) = 0{,}1\%$ et pour $d(W) = 1\%$, le gain positif est obtenu pour des matrices ayant respectivement 9.000 et 2.000 colonnes. Les gains en mémoire dépassent le surplus en complexité uniquement pour les densités $d(W) = 0{,}1\%$ et $d(W) = 1\%$ à partir des matrices ayant respectivement 3.000 et 16.000 colonnes. Cela signifie que plus la densité et la taille de matrice sont grandes, plus le gain et l'intérêt pour le mode CSC-G sont garantis. Le gain maximal en terme de coût mémoire est de 44% avec $d(W) = 1\%$. La différence maximale entre le gain et le surplus en complexité peut atteindre 40% pour $d(W) = 1\%$.

Pour de grandes matrices (> 100.000 lignes) et pour une densité $d(W) = 0{,}01\%$, le gain en mémoire devient positifs à partir des matrices ayant 600.000 colonnes. Celui-ci dépasse le surplus en complexité à partir des matrices ayant 1.110.000 colonnes. Cette figure montre aussi que le gain positif en coût de mémorisation est toujours réalisé pour les densités $d(W) = 0{,}1\%$ et $d(W) = 1\%$. Le dépassement de la complexité est presque toujours assuré (à partir de 200.000 colonnes pour $d(W) = 0{,}1\%$). Le gain maximal en terme de coût mémoire est de 42% avec $d(W) = 1\%$. La différence maximale entre le gain et le surplus en complexité peut atteindre 40% pour $d(W) = 0{,}1\%$ et pour $d(W) = 1\%$.

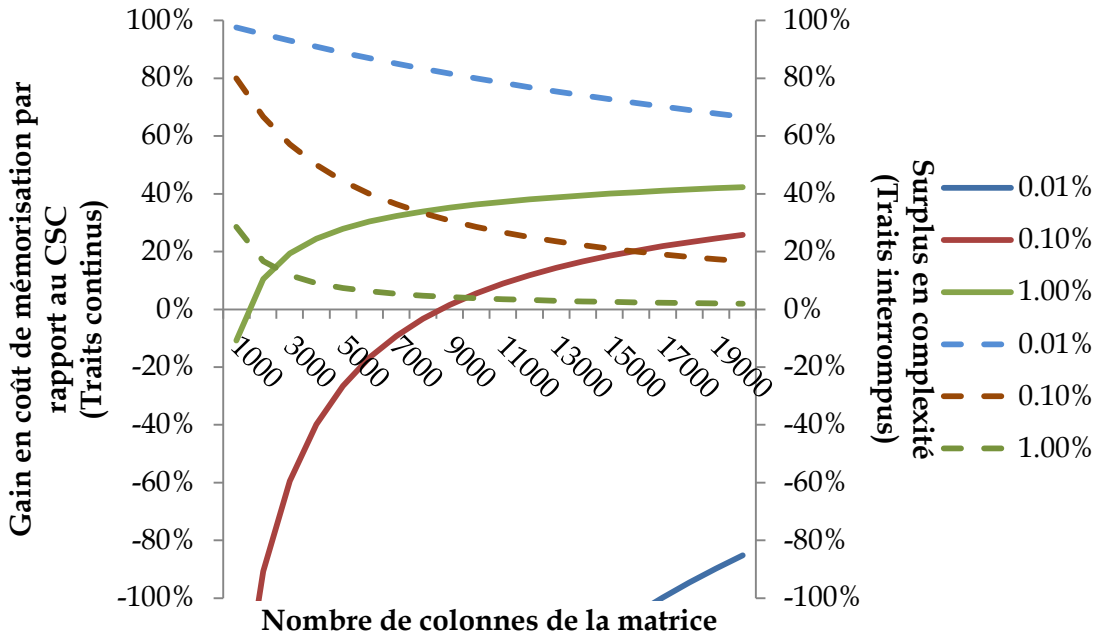


Figure VI.1 Gain en coût de mémorisation (courbes en traits continus) et surplus en termes de complexité temporelle (courbes en traits interrompus) du CSC-G par rapport au CSC: cas de petites matrices

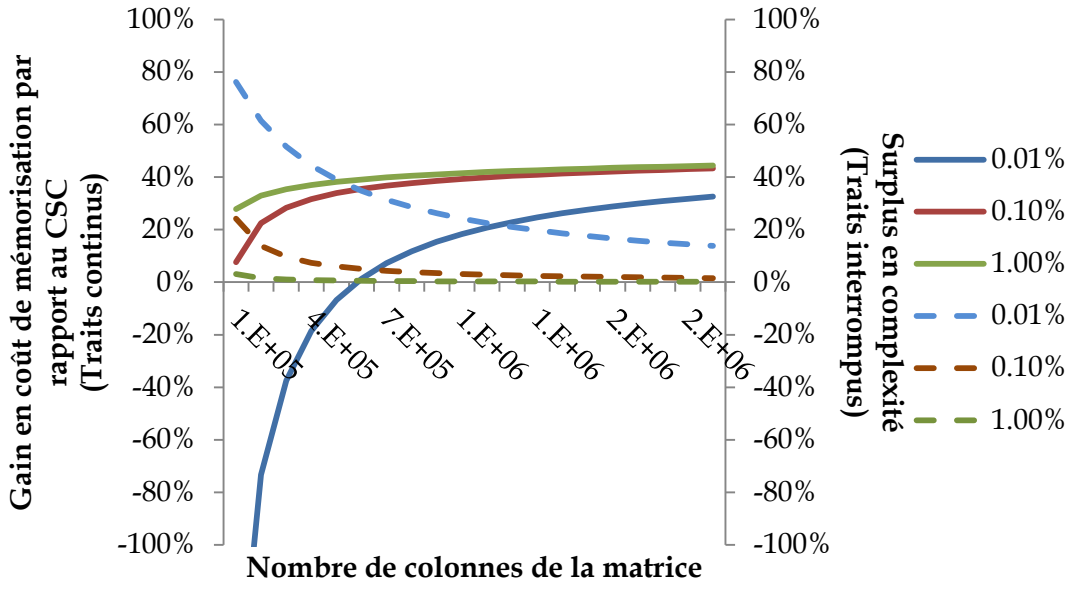


Figure VI.2 Gain en coût de mémorisation (courbes en traits continus) et surplus en termes de complexité temporelle (courbes en traits interrompus) du CSC-G par rapport au CSC: cas de grandes matrices

Les gains en coût de mémorisation du mode CSC-G en intégrant le stockage de données codées sur 16, 32 et 64 bits ont aussi été évalués. La tendance est identique avec des gains qui augmentent en fonction de la taille de la matrice et en fonction de sa densité. Cependant les gains sont plus faibles. Les résultats montrent des gains atteignant respectivement 25%, 18% et 11% pour de grandes matrices ayant une densité de 1%.

En somme, l'intérêt du mode CSC-G dépend de la taille de la matrice et de sa densité comme l'indique l'expression (58). Plus la densité et la taille de matrice sont élevées, plus le gain en coût de mémorisation est certain et le surplus en complexité temporelle devient négligeable par rapport à ce gain. Cependant, il faut noter que ce gain dépend du choix de b. Sa valeur minimale peut être facilement déterminée à partir de l'équation (58) si jamais la taille de la matrice et sa densité sont connues.

C. Architecture générique des réseaux à clones (ARCH00)

L'objectif de cette section est de décrire une première architecture générique mettant en œuvre de réseaux à clones. Les modèles présentés dans le CHAPITRE V (réseaux à clones, SNSM, DNDM, …) sont supportés. La matrice d'adjacence supportée est fournie en format CSC ou CSC-G.

1. Notation architecturale

La Figure VI.3 introduit les notations principales qui seront utilisées par la suite et peut servir de référence. Ces notations et leur définition seront rappelées au fur et à mesure.

Notations :	
Nt	Nombre total de clones dans le réseau manipulé
d	Densité de ce réseau
Mca	Mémoire contenant la liste des clones actifs
Mna	Mémoire contenant la liste des neurones actifs
MM	Mémoire contenant la matrice sous format CSC ou CSC-G
Nca	Nombre de clones actifs contenus dans la mémoire Mca
$Nlig$	Nombre moyen de lignes associés à chaque colonne de la matrice d'adjacence
$La(Mem)$ ou $La(Reg)$	Largeur d'une mémoire Mem ou d'un registre Reg
$\|Mem\|$ ou $\|Reg\|$	Profondeur d'une mémoire Mem ou d'un registre Reg
PE	Processing Element : élément de calcul
$\|PE\|$	Nombre d'éléments de calcul de l'architecture

Figure VI.3 Notations principales utilisées pour présenter de l'architecture générique

2. Architecture générique : composants principaux et fonctionnement général

La Figure VI.4 montre un schéma de l'architecture proposée. Elle intègre les composants suivants :

- La mémoire *MM* (Mémoire Matrice) contenant la matrice parcimonieuse sous format CSC ou CSC-G (Stockée dans la zone *pointeurs* et *lignes*). Cette mémoire peut être externe ou interne à la plateforme visée (FPGA ou ASIC). Si elle est externe, alors elle

doit être chargée par le contrôleur mémoire. Elle contient aussi l'association des neurones aux clones (Zone *asn : Associated Neuron, neurone associé*).

- La mémoire *Mca* contenant la liste des clones actifs. Cette liste est utilisée pour parcourir la matrice et pour déterminer les contributions des clusters du réseau pour chacun de ses différents clones. Elle doit être mise à jour à la fin de chaque itération de décodage. En effet, le décodage s'effectue en plusieurs itérations et chaque itération prend en compte la liste des clones actifs obtenue à l'itération précédente.
- La mémoire *Mna* contenant la liste des neurones actifs. Cette liste est retournée vers l'extérieur de l'architecture après le décodage.
- Les éléments de calcul *PE* (*Processing Element*) contenant les informations de leurs clones associés (présence d'une mémoire interne) ainsi que les ressources nécessaires pour calculer leur score et mettre à jour leur valeur.
- Le Contrôleur Général (*CG*) contrôlant la totalité du calcul (routage des données provenant des mémoires en direction d'un unique *PE*, calcul du maximum du réseau, tests des valeurs des clones afin de retrouver les clones actifs, etc.).

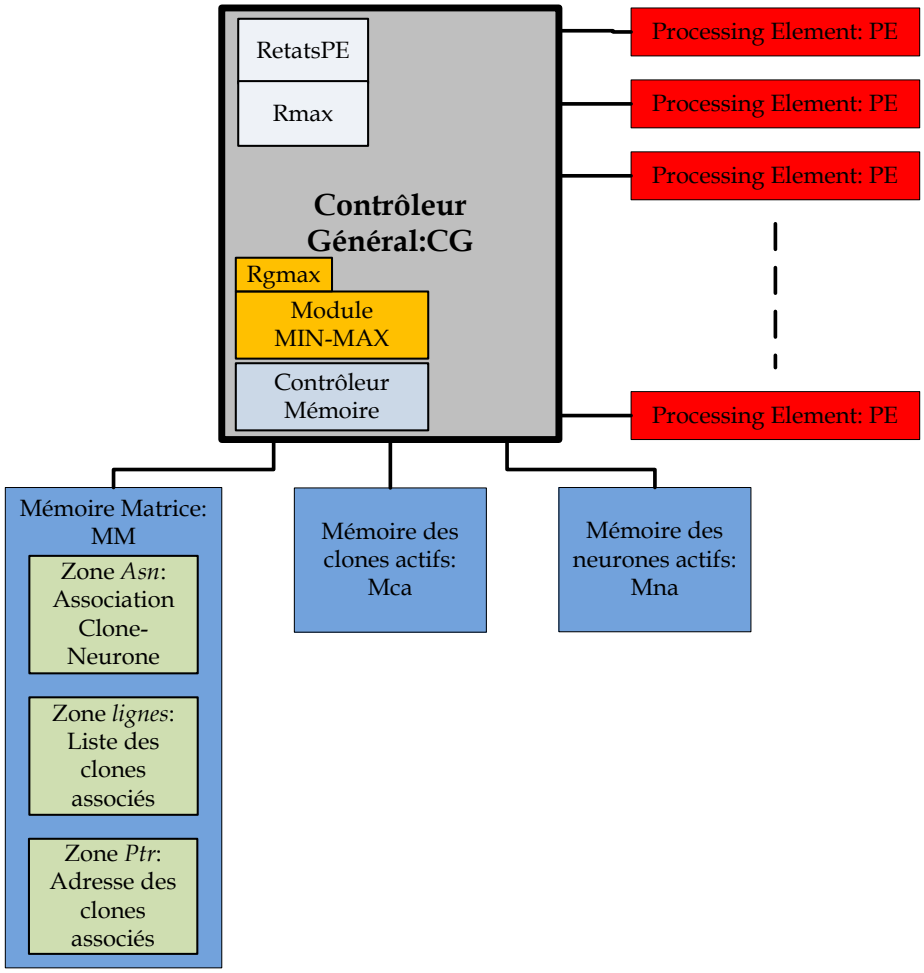


Figure VI.4 Schéma de l'architecture générique de réseau à clones

Au début du décodage, des neurones sont fournis comme entrées de l'architecture. Ces neurones sont actifs. Pour chaque neurone lu, ses clones associés sont stockés dans la mémoire *Mca* contenant les clones actifs. Cette tâche est réalisée en lisant la mémoire *MM* qui contient l'association de chaque neurone à ses clones.

La Figure VI.5 montre les cinq étapes d'une itération de décodage :

- *INITialisation (INIT)* : cette étape consiste à initialiser les contributions des clusters pour chaque clone associé ainsi que les valeurs des clones (Initialisation des mémoires internes ainsi que des registres dans les *PEs*).
- *PRE-Calcul (PREC)* : cette étape consiste à parcourir les clones actifs dans la mémoire *Mca* et pour chaque clone actif : (1) sa valeur est initialisée dans son *PE* associé et (2) ses clones connectés sont lus (lecture de la matrice dans MM) et traités. Pour chaque clone connecté, la contribution du cluster actif en cours (cluster contenant le clone actif en cours) pour ce clone connecté est mis à 1. Ainsi, la fin de cette phase, les espaces alloués aux clones contiennent les contributions qui leur sont dédiées et qui permettront ensuite de calculer leur score.
- *CACUL de score (CALC) :* pour chaque *PE*, cette étape consiste à : (1) calculer le score de chaque clone associé au *PE*, (2) déterminer le score maximal de chaque *PE* (score local) ainsi que le score maximal du réseau. Le score d'un clone est obtenu en sommant les contributions provenant des clusters distants ainsi que sa valeur. Le score maximal du réseau est obtenu en calculant le maximum parmi les scores locaux fournis par les *PEs*.
- *Winner Take All (WTA)* : pour chaque *PE,* cette étape consiste à mettre à jour l'état de chaque clone associé en comparant son score au score maximal du réseau.
- *Mise A Jour (MAJ)* : cette étape consiste à mettre à jour la liste des clones actifs qui se trouve dans la mémoire *Mca*. Cela revient à parcourir les valeurs des clones stockées dans chaque *PE*, à évaluer leur état et à les ajouter à la mémoire *Mca* afin de mettre à jour la liste de clones actifs. Cette liste est nécessaire pour réaliser la prochaine itération du décodage.

Pendant la phase de mise à jour de la liste des clones actifs MAJ, les neurones associés aux clones actifs peuvent être lus progressivement dans la mémoire MM afin d'obtenir la liste des neurones actifs du réseau.

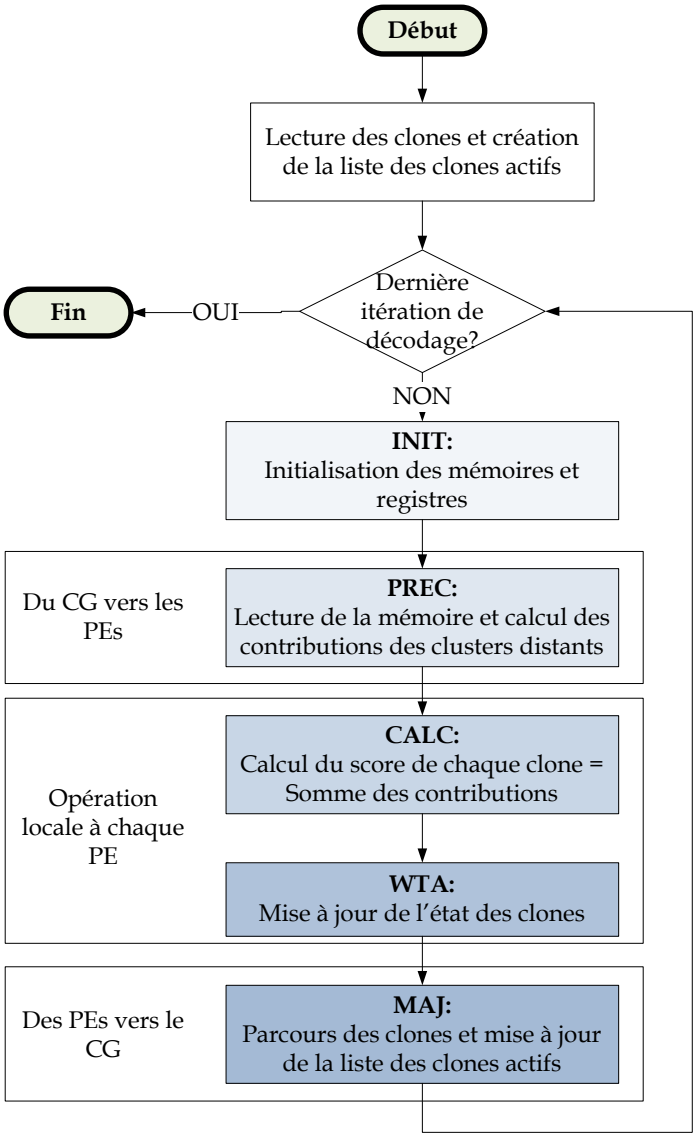


Figure VI.5 Déroulement du décodage

3. Identification des clones dans le réseau et dans l'architecture

A chaque clone est associé un identifiant *Idcl* unique dans le réseau. Cet identifiant correspond à un numéro de ligne et de colonne de la matrice d'adjacence de ce réseau.

L'identifiant d'un clone est une valeur composée de trois parties :

- L'identifiant du sous-réseau auquel appartient le clone ;
- L'identifiant du cluster auquel appartient le clone : cet identifiant est relatif au sous-réseau auquel le cluster appartient ;
- L'identifiant du clone lui-même relativement au cluster.

La Figure VI.6 représente la matrice d'adjacence d'un DNDM(4,3,4,4) contenant 4 sous-réseaux (SR0-SR3), chacun ayant 3 clusters (0-2) avec chaque cluster comportant 4 clones et

un clone en moyenne étant alloué par neurone. Chaque clone a un identifiant relatif à son cluster (0-3). Un clone peut être alors défini par 0_2_0 qui signifie : clone 0 appartenant au cluster 2 du sous-réseau 0 (SR0). Son numéro de colonne/ligne associé est 8. Les cases blanches indiquent que la connexion entre les clones est possible et les cases sombres indiquent que la connexion est impossible.

Numéros de lignes			Sous-réseaux	SR0			SR1			SR2			SR3		
			Clusters	0	1	2	0	1	2	0	1	2	0	1	2
	Sous-réseaux	Clusters	Clones	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3	0-3
0-3	SR0	0	0-3												
4-7		1	0-3												
8-11		2	0-3												
12-15	SR1	0	0-3												
16-19		1	0-3												
20-23		2	0-3												
24-27	SR2	0	0-3												
28-31		1	0-3												
32-35		2	0-3												
36-39	SR3	0	0-3												
40-43		1	0-3												
44-48		2	0-3												

Figure VI.6 Identification des clones: Association des numéros de colonnes/lignes aux clones d'un réseau DNDM(4,3,4,4)

4. Description détaillée des composants de l'architecture

a) Le Contrôleur Général (CG)

Le *CG* contient les modules suivants :

- Un contrôleur mémoire pour gérer les accès aux mémoires *MM*, *Mca* et *Mna*. Si ces mémoires sont externes à la plateforme, son rôle sera de charger et de sauvegarder le contenu de ces mémoires au fur et à mesure de leur lecture. Dans le cas de *MM*, il s'agira de charger les colonnes (clones locaux) associés aux lignes dans *Mca* (clones actifs). Dans le cas de *Mca*, il s'agira de charger les clones actifs au fur et à mesure. Dans le cas de *Mna*, il s'agira de charger les neurones actifs au début du décodage et de sauvegarder les neurones actifs à la fin du décodage.
- Un module *MIN-MAX* permettant de calculer le minimum ou le maximum des scores provenant des *PEs*. Ce module permet d'évaluer le score maximal du réseau. Il contient un arbre de comparateurs.
- Un registre *Rgmax* (Register-Global Maximum : Registre-Maximum Global) contient le score maximal du réseau. Ce registre est mis à jour par le module *MIN-MAX*.

- Un registre *Rmax* dont chaque mot reçoit le score maximal provenant d'un unique *PE.*
- Un registre *Retats* dont chaque mot reçoit un unique bit de chaque *PE* (la valeur d'un unique clone).

b) La Mémoire Matrice (MM)

La mémoire *MM* contient 3 zones logiques :

- La zone *ptr* (*Pointeur*) contient pour chaque colonne de la matrice W (chaque clone distant) l'adresse de la zone qui contient l'ensemble des lignes (clones) contenant une valeur non nulle (adresses de la liste des clones auxquels il est connecté). *ptr* contient donc $Nt + 1$ mots mémoire dans le cas d'un DNDM(K,C,L,Ncl), un mot pour chaque clone du réseau et un mot supplémentaire contenant le nombre d'éléments non nuls dans la matrice (égal au nombre total d'éléments à 1 dans cette matrice).
- La zone *asn* (*ASsociated Neurons : neurones associés*) contient Nt mots, chaque mot étant associé à un unique clone du réseau et contenant le neurone auquel il est associé. Cette représentation peut être associée à une autre qui indique pour chaque neurone ses clones associés. Cette dernière est utile car lors de l'initialisation du décodage, les neurones issues de l'entrée doivent être associés à des clones afin d'obtenir la liste des clones actifs. De même, à la fin du décodage, les neurones associés aux clones actifs doivent être récupérés afin d'obtenir la liste des neurones actifs.
- La zone *lignes* contient dans chaque mot un numéro de ligne (resp. un clone) associé (resp. connecté) à une colonne (à un clone distant) indiquée dans *ptr*. Sa profondeur est égale au nombre maximal de poids synaptiques alloués à la matrice d'adjacence du réseau.

Note :

Dans les mémoires présentées, l'identifiant d'un clone occupe un unique mot mémoire. Cependant, cet identifiant peut en occuper plusieurs. Cela permet d'identifier un plus grand nombre de clones tout en maintenant une faible largeur mémoire. Dans ce cas, la lecture d'un identifiant prend plusieurs cycles. L'inverse est aussi possible et un identifiant peut occuper une partie du mot-mémoire. Ainsi, plusieurs identifiants peuvent être obtenus avec une seule lecture de la mémoire.

c) Les éléments de calcul (PE)

Chaque PE renferme les éléments suivants :

- Une mémoire interne *Mint* contenant les informations des clones associés à ce *PE*;
- Un registre *Retats* contenant les valeurs des clones ;

- Un registre *Rlmax* (Register-Local Maximum : Registre-Maximum Local) contenant le score maximal des clones associés à ce *PE*. Il est constitué d'un unique mot-mémoire. Sa largeur est égale à celle de *Mint* ($La(Rlmax) = La(Mint)$) ;
- Un module *Tadd* (Tree adder : Arbre d'additionneurs) permettant de calculer le score d'un clone. Il est constitué d'un arbre d'additionneurs. Cet arbre prend en entrée un mot mémoire (vecteur de bits), additionne les bits qui le composent et retourne un entier.
- Un registre *Rscore* cumulant les valeurs provenant de *Tadd* et servant au calcul du score des clones. Il contient un unique mot mémoire et a une largeur égale à celle de *Mint*. ($La(Rscore) = La(Mint)$).

La Figure VI.7 donne l'aperçu d'un *PE.*

Chaque *PE* est associé à un ensemble de lignes de W équivalent à un ensemble de clones du réseau. Le *PE* associé à un clone va contenir toutes les informations associées à ce clone (Les contributions des clusters distants, son score, sa valeur, …). Soit $(PE_i)_{0\leq i\leq |PE|-1}$ la liste des *PEs* existants dans l'architecture. Alors pour deux clones identifiés par $Idcl1$ et $Idcl2$ avec $Idcl2 = Idcl1 + 1,$ si $Idcl1$ est associé à PE_{Idcl1} alors $Idcl2$ est associé à $PE_{(Idcl1+1)\ mod\ |PE|}$. Par exemple, si l'architecture contient 2 *PEs* (c.à.d. $|PE| = 2$), les clones identifiés par 0, 1, 2 seront respectivement associés aux PE_0, PE_1, PE_0.

La mémoire *Mint* d'un *PE* contient les informations de plusieurs clones. Pour un clone en particulier, elle stocke : la contribution de chaque cluster distant ainsi que son score. Ainsi, cette mémoire est divisée en plusieurs zones logiques (Voir Figure VI.7), chacune étant associée à un unique clone. Une zone logique est composée de deux parties :

- La première partie $Sccl$ (SCore CLone) contient les contributions dues aux clusters distants. Chaque bit de cette zone est associé à un unique cluster distant et indique si le clone est connecté à au moins un clone actif de ce cluster (1) ou non (0). Un bit supplémentaire est ajouté pour stocker la valeur du clone. Le nombre total de bits de $Sccl$ doit être supérieur égal au nombre de clusters dans le réseau décodé, c.à.d. C bits. Ainsi, le nombre de mots $|Sccl|$ de $Sccl$ dépend de C et de la largeur de la mémoire *Mint* $La(Mint)$ tel que: $|Sccl| * La(Mint) \geq C$.
- La seconde partie Sc consiste en un unique mot mémoire contenant le score du clone.

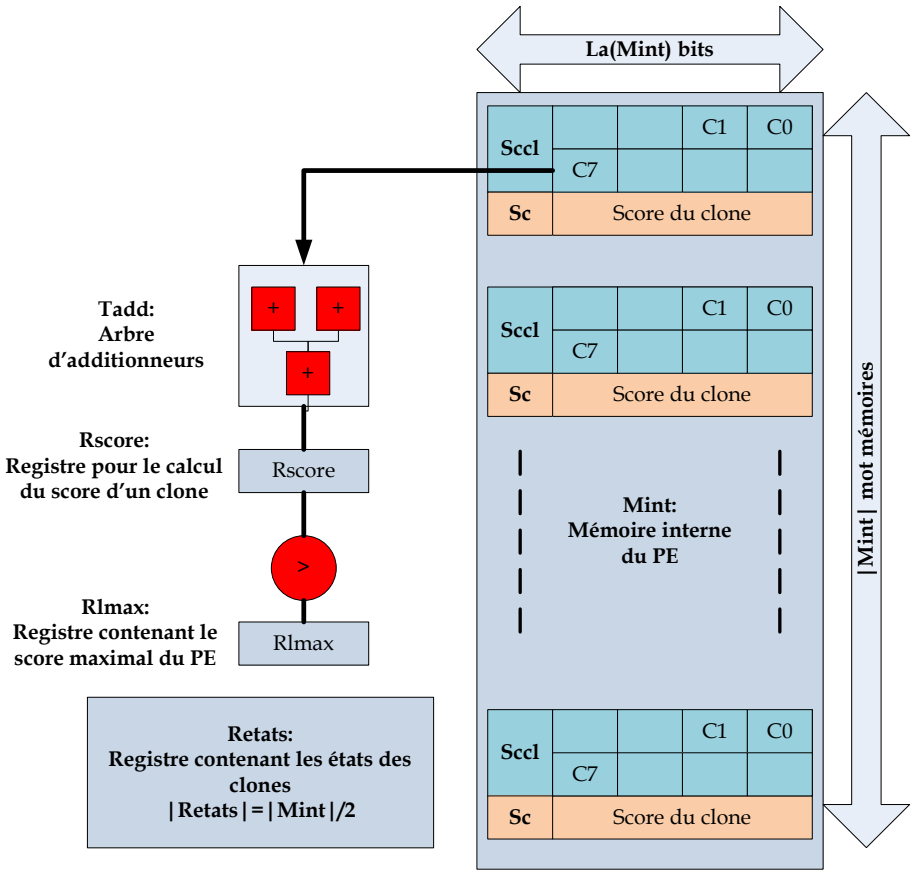


Figure VI.7 Schéma d'un *PE*

Note 1 : Calcul du score d'un clone

Selon cette disposition, le score d'un clone est égal à la somme des bits de $Sccl$ (C.à.d. la somme des contributions de chaque cluster additionné à la valeur du clone).

Le nombre de mots contenus par une zone logique est alors égal à $(|Sccl| + 1)$.

Exemple : Pour un réseau DNDM(4,3,3,3), si la largeur mémoire $La(Mint) = 16\ bits$ alors la profondeur de la zone logique allouée à un clone est $|Sccl| + 1 = 1 + 1 = 2$. En effet, un mot mémoire $(16bits > 3\ clusters + 1)$ est suffisant pour stocker les contributions de tous les clusters.

Le nombre maximal de zones logiques que peut contenir la mémoire *Mint* est de $\left\lfloor \frac{|Mint|}{2} \right\rfloor$. En effet, chaque zone logique contient au minimum deux mots. Ainsi, le registre *Retats* contient $\left\lfloor \frac{|Mint|}{2} \right\rfloor$ bits avec un bit pour chaque clone associé au *PE*.

Pour mettre à jour la contribution d'un cluster par rapport à un clone, il faut connaître le *PE* associé à ce clone, l'adresse de sa zone logique et le bit à modifier dans sa partie $Sccl$. Les détails de ces opérations sont donnés en annexe (Voir Annexe A).

5. Condition de décodage sans stockage externe (CDSE)

Afin que le décodage d'un message par un réseau soit possible, il faut que l'architecture dispose de suffisamment de zones logiques. Si la mémoire disponible est insuffisante, alors des étapes de sauvegarde et de chargement des zones logiques des clones à partir d'une mémoire externe plus large sont nécessaires.

Définition : Condition de décodage sans stockage externe

> Une architecture définie par son nombre d'éléments de calcul $|PE|$ et par la configuration de sa mémoire interne (largeur $La(Mint)$ et profondeur $|Mint|$) peut réaliser le décodage sans stockage externe pour un réseau si et seulement si l'ensemble des mémoires internes des *PEs* contiennent suffisamment d'espace pour stocker les zones logiques de tous les clones de ce réseau.

Pour un DNDM(K,C,L,Ncl), il faut Nt zones logiques.

Ainsi, pour une architecture définie par ses éléments de calcul et par sa configuration de sa mémoire interne, la condition CDSE pour un réseau DNDM(K,C,L,Ncl) est donnée par :

$$Nt \leq |PE| * \left\lfloor \frac{|Mint|}{(|Sccl|+1)} \right\rfloor \ zones\ logiques \tag{59}$$

L'expression $\left\lfloor \frac{|Mint|}{(|Sccl|+1)} \right\rfloor$ représente le nombre total de zones logiques dans la mémoire interne des *PEs*. La multiplication par $|PE|$ permet d'obtenir le nombre maximal de zones logiques que l'architecture peut manipuler par rapport au nombre de clusters par sous-réseau *C*.

Par exemple, considérons la configuration du réseau utilisée dans [104] qui est un GBNN(8,512). Ce réseau est équivalent à un DNDM(1,8,512,512). Pour une configuration de l'architecture définie par $|PE| = 64, La(Mint) = 16$ bits et $|Mint| = 256$ mots mémoire. Chaque zone logique est constituée de 2 mots mémoire (1 mot mémoire pour la partie *Sccl* et + 1 pour obtenir la profondeur totale). La condition donne alors $Nt = 4096 \leq 8192$ zones logiques. D'où cette architecture est suffisante pour traiter un tel réseau.

6. Configuration initiale

Avant de pouvoir manipuler un réseau, l'architecture doit se configurer en fonction des caractéristiques de ce réseau: le nombre de réseau *C* et le nombre total de clones dans le réseau Nt. La configuration consiste en deux calculs :

- Calcul du nombre de mots mémoire $|Sccl|$ de la partie $Sccl$ de chaque zone logique. Ce calcul permet de déterminer le nombre de mots présents dans la zone logique de chaque clone $(|Sccl| + 1)$;

- Calcul du nombre de clones du réseau associé à chaque élément de calcul afin de ne manipuler que les zones logiques associés aux clones du réseau.

7. Description détaillée du décodage

Dans cette section, les étapes du décodage sont progressivement détaillées afin de caractériser le temps de calcul de l'architecture pour une itération de décodage.

Définition : Temps de calcul ou latence de l'architecture

> Le temps de calcul est une estimation du nombre total de cycles que l'architecture consomme pour réaliser la tâche qui lui a été assignée.

Le temps de calcul mesuré ici se réfère au nombre de cycles requis pour réaliser une itération de décodage. Il s'agit d'une complexité temporelle.

a) INIT

L'initialisation du décodage commence par l'initialisation des mémoires internes et des registres des *PEs*. Pour chaque zone logique associée à un clone, la partie $Sccl$ de la zone et le score du clone sont initialisés à 0. De même, les registres *Rlmax* et *Retats* des *PEs* sont initialisés à 0.

La latence T_{INIT} pour cette opération égale au temps mis pour initialiser toutes les zones logiques utilisées par le réseau (Zone *Sccl* et score) :

$$T_{INIT} = \frac{(|Sccl| + 1) * Nt}{|PE|} \text{ cycles} \tag{60}$$

En effet, chaque élément de calcul contient en moyenne $\frac{Nt}{|PE|}$ zones et chaque zone contient $|Sccl| + 1$ mots mémoire.

b) PREC

Cette phase consiste à déterminer les contributions des clusters distants pour chaque clone. Ainsi, pour chaque clone actif *Idcl2* lu dans la mémoire des clones actifs *Mca* (c.à.d. une colonne de la matrice), les opérations suivantes sont réalisées :

- Détermination de l'adresse du bit à modifier dans les zones logiques des clones (Adresse dans laquelle doit être inscrite la contribution du cluster contenant le clone actif) ;
- Détermination de l'adresse de la zone logique et du *PE* associé à *Idcl2* (Voir Annexe A) ;
- Mise à 1 du bit pour le clone *Idcl2* et à l'adresse calculée. Cette mise à 1 sert à initialiser l'état de ce clone ;

- Lecture des clones (lignes) connectés (contenant une valeur non nulle avec) à *Idcl2* ;
- Pour chaque clone lu *Idcl1* :
    - Détermination de l'adresse de sa zone logique et de son *PE* associé (Voir Annexe A);
    - Mise à 1 du bit dans la zone logique de ce clone, indiquant ainsi qu'il a reçu une contribution provenant du cluster contenant le clone actif en cours *Idcl2* .

La latence de chaque *PE* pour la mise à jour d'un unique clone est de 4 cycles : (1) lecture d'un mot mémoire (2 cycles), (2) modification du bit lié à la contribution (1 cycle) et (3) stockage (1 cycle).

La latence totale de la phase PREC est proportionnelle au nombre de clones actifs Nca ainsi qu'au nombre moyen de lignes de la matrice *W* contenant des valeurs non nulles, $Nlig$. $Nlig$ dépend de la densité de matrice et est égale à $d(W) * Nt$.

Le temps maximal T_{PREC} mis pour réaliser cette phase est égal à :

$$T_{PREC} = T_{CG} * Nlig * Nca \text{ cycles} \quad (61)$$

Avec $T_{CG} = 4$, le temps de calcul du *CG* : (1) lecture de la mémoire (2 cycles), (2) détermination du *PE* associé à *Idcl,* de l'adresse du bit à mettre à jour et de l'adresse de la zone logique associé à *Idcl*, (3) transmission des données vers le *PE* associé à *Idcl*. Nca est le nombre de clones actifs dans la mémoire *Mca*.

Note : Cas de l'usage du mode CSC-G

> Si la matrice utilise le mode de représentation CSC-*C*, alors la valeur $Idcl1$ lue sera définie relativement à son bloc. Cette valeur devra être décodée afin de correspondre à sa véritable valeur (Voir CHAPITRE VI.B.1).

c) CALC

Cette étape est locale à chaque *PE*. Le score de chaque clone est calculé en deux phases :

- Initialisation d'un registre *Rscore* à 0 ;
- Somme des contributions. Pour réaliser cette phase, les mots mémoire de la partie *Sccl* sont lus et pour chaque mot lu, la somme de ses bits est calculée grâce à l'arbre d'additionneur *Tadd* et est cumulée à la valeur contenue dans *Rscore*.

Après avoir calculé le score d'un clone, celui-ci est sauvegardé dans la zone logique du clone. D'un autre côté, il est aussi comparé au maximum local du *PE* qui est contenu dans son registre *Rlmax*. Si ce score est supérieur à la valeur dans *Rlmax,* alors son contenu est remplacé par le nouveau score.

Lorsque tous les *PEs* ont fini de calculer, le score présent dans chaque *Rlmax* est envoyé vers le module *MIN-MAX* du contrôleur CG afin que celui-ci calcule le score maximal du réseau (Opération de MAX). La valeur obtenue par *MIN-MAX* est alors diffusée vers les *PEs* en vue de la phase de la mise à jour de l'état des clones (MAJ).

Le temps de traitement d'un unique clone est de $|Sccl| + 3$ cycles : (1) initialisation de *Rscore* : (1 cycle), (2) calcul du score du clone incluant lecture et calcul ($|Sccl|$ cycles), (3) comparaison à *Rlmax* / Stockage du score du clone (1 cycle), (4) mise à jour de *Rlmax* (1 cycle). Cette phase dure alors en moyenne :

$$T_{CALC} = (|Sccl| + 3) * \frac{Nt}{|PE|} \text{ cycles} \quad (62)$$

En effet, $\frac{Nt}{|PE|}$ clones du réseau en moyenne sont associés à chaque *PE*.

d) WTA

Dans cette étape, chaque *PE* parcourt les zones logiques se trouvant dans sa mémoire interne. Pour chaque zone parcourue, il compare le score du clone associé à la zone au score maximal du réseau. Ensuite, il met à jour son état s'il a un score égal à ce score. La latence de l'évaluation d'un clone est de 4 cycles : (1) lecture du score du clone (2 cycles), (2) comparaison au max et (3) mise à jour de l'état du clone. L'évaluation de tous les clones prend alors :

$$T_{WTA} = 4 * \frac{Nt}{|PE|} \text{ cycles} \quad (63)$$

e) MAJ

L'objectif de cette phase est de mettre à jour la liste des clones. Cette phase consiste à parcourir les clones dans tous les éléments de calcul *PE*, à évaluer leur état et à les ajouter dans la mémoire *Mca* si jamais leur état est égal à 1.

$Idcl$ étant un identifiant de clone initialisé à 0, cette opération se déroule de la manière suivante :

- Pour chaque clone auquel il est associé, chaque *PE* émet vers le contrôleur *CG*, l'état de ce clone se trouvant dans *Retats.* ;
- Lorsque le contrôleur a reçu les informations de chaque *PE*, il parcourt chaque état et évalue si la valeur est 1 ou non. Si elle est égale à 1, le clone considéré est actif. Dans ce cas, la valeur $Idcl$ est ajoutée à la mémoire *Mca* car $Idcl$ représente l'identifiant du clone traité. A chaque passage à une ligne suivante, $Idcl$ est incrémenté de 1 ;

- A la fin du parcours de *RetatsPE*, s'il reste des clones à évaluer (c.à.d. $Idcl < Nt$), le processus recommence avec la zone logique suivante pour chaque mémoire interne de chaque *PE*.

Le temps total T_{MAJ} mis par le contrôleur *CG* pour évaluer tous les clones est égal à :

$$T_{MAJ} = 3 * Nt \text{ cycles} \tag{64}$$

3 représente 3 cycles correspondant à : (1) lecture d'un mot de *RetatsPE* (2 cycles), (2) vérification de l'état du clone et stockage dans la mémoire *Mca* si valeur à 1. T_{MAJ} est le temps mis pour créer la liste des clones actifs.

Le temps de calcul global Tc pour une itération de décodage est alors égal à :

$$Tc = \frac{(|Sccl| + 1) * Nt}{|PE|} + T_{CG} * Nlig * Nca + (|Sccl| + 7) * \frac{Nt}{|PE|} + 3 * Nt \text{ cycles} \tag{65}$$

Note : Cas d'un réseau ayant une organisation hétéro-associative

Un réseau « hétéro-associatif » à clusters apprend l'association entre deux messages $\boldsymbol{m_0}$ et $\boldsymbol{m_1}$. Il est capable de trouver $\boldsymbol{m_1}$ à partir de $\boldsymbol{m_0}$. Si le contraire est aussi possible le réseau est bidirectionnel.

Un tel réseau est muni de deux couches $L^{(0)}$ et $L^{(1)}$. Chaque couche contient des clusters. Un clone contenu dans un cluster et appartenant à une couche ne peut être connecté qu'aux clones faisant parti des clusters de l'autre couche.

A l'apprentissage, pour chaque couche, un clone actif est sélectionné dans chaque cluster d'un unique sous-réseau. L'apprentissage est réalisé en mettant à 1 les connexions entre les clones sélectionnés.

Au décodage, des messages bruités $\widetilde{\boldsymbol{m_0}}$et $\widetilde{\boldsymbol{m_1}}$ sont reçus. Les clones sont alors activés en fonction de la valeur de chaque symbole contenu dans les messages. Le décodage consiste à calculer la valeur de chaque clone. Les valeurs des clones de $L^{(1)}$ sont calculées en fonction des valeurs des clones de $L^{(0)}$ et vice versa. Ainsi, afin que l'architecture puisse calculer les valeurs des clones de $L^{(1)}$, il faut que les clones actifs de $L^{(1)}$ soient lus avant les clones de $L^{(0)}$. En effet, tandis que la lecture des clones de $L^{(1)}$ sert uniquement à initialiser les valeurs de ces clones, la lecture des clones de $L^{(0)}$ sert à calculer le score ainsi que la valeur des clones de $L^{(1)}$.

D. Améliorations de l'architecture

Cette section présente cinq améliorations permettant de réduire :

- La latence pour réaliser une itération de décodage ;
- Le coût mémoire requis pour réaliser un décodage ;
- Le coût en ressources de calcul de l'architecture.

Ces améliorations peuvent se combiner et sont les suivantes :

(1) Fusion des phases de décodage (ARCH01, Voir Annexe B) : cette amélioration consiste à réaliser le calcul de score (Phase CALC) directement lors de la mise à jour la zone logique d'un clone (Phase PREC). Cela permet de réduire légèrement le temps de calcul global. Elle a aussi pour conséquence le remplacement du module *Tadd* (arbre d'additionneurs) présent dans les *PEs* par un simple additionneur, ce qui implique aussi un gain en ressources de calcul. En effet, lors de la mise à jour de la zone logique d'un clone, le score est incrémenté de 1 uniquement si le bit modifié passe de 0 à 1. Cela revient à prendre en compte de la contribution d'un cluster distant pour ce clone une et une seule fois.

(2) Stabilisation de la profondeur des zones logiques des clones (ARCH02, Voir Annexe C) : cette amélioration élimine la dépendance de la profondeur des zones logiques par rapport au nombre de clusters par sous-réseau. En effet, le parcours des clones actifs est équivalent à un parcours des clusters contenant ces clones. Si ces clones sont classés par cluster, la lecture de ces clones revient à parcourir les clusters au fur et à mesure et à déterminer les contributions de chaque cluster actif lu pour chaque clone. Dans ce cas, le score d'un clone est incrémenté en fonction des clusters parcourus. Lors de la mise à jour du score d'un clone *Idcl1*, si le cluster contenant le clone actif *Idcl2* en cours est différent du dernier cluster ayant mis à jour le score de *Idcl1*, alors le score est incrémenté de 1. Ainsi, la partie *Sccl* des zones logiques disparait et leur profondeur devient constante.

(3) Allocation dynamique des zones logiques (ARCH03, Voir Annexe C) : dans l'architecture présentée, une zone logique est associée à chaque clone existant dans le réseau. Cependant, lors du décodage, seules les zones logiques des clones ayant un poids à 1 avec les clones actifs subissent un changement de score. Afin d'utiliser moins de mémoire lors du décodage, un mécanisme d'allocation dynamique des zones logiques est introduit. Il consiste à allouer des zones logiques uniquement aux clones qui subissent une mise à jour de leur score. Cela se traduit par l'usage de mémoires (une mémoire par *PE*) stockant pour chaque clone associé à ce *PE* l'adresse de sa zone logique.

(4) Elimination de l'arbre de comparateurs (ARCH04, Voir Annexe D) : le module *MIN-MAX* du *CG* utilise des ressources dont le coût augmente en fonction du nombre d'éléments de calcul (les *PE*s). Lorsque le contrôleur ordonne la mise à jour du score d'un clone associé à un *PE* et

sachant que le score maximal du *PE* est aussi mis à jour dans cette opération, le contrôleur peut vérifier après un délai constant si ce score provoque la mise à jour du score maximal du réseau. Tandis que le contrôleur stocke de manière continue les numéros des *PEs* à scruter dans une boîte de message, un autre module se charge de lire ces numéros et de lire le score associé. Ainsi, en scrutant les *PEs* au fur et à mesure, le score maximal du réseau est mis à jour grâce à un unique comparateur.

(5) Construction dynamique de la liste des clones actifs (ARCH05, Voir Annexe D) : cette amélioration se base sur la précédente et consiste à créer la liste des clones actifs directement lors de la phase de calcul de score PREC. Pour cela, deux mémoires supplémentaires sont ajoutées au contrôleur : une contenant la liste temporaire des clones actifs *Mca*2, l'autre contenant les clones dont les PEs ont été récemment mis à jour *Mcl*. Si jamais un *PE* scruté dans la boîte de message fournit un score qui a changé et qui est devenu supérieur ou égal au score maximal du réseau, et sachant que le clone associé à ce *PE* et propriétaire de ce score est connu (car stocké dans *Mcl*), alors *Mca2* doit être mise à jour. Si le score est supérieur, la liste est réinitialisée et ce clone doit être ajouté à cette liste car les clones présents dans cette liste ont un score inférieur au sien. Si le score est égal, le clone est directement ajouté à la liste et si le score est inférieur, le clone est ignoré. Ainsi, cette amélioration permet d'éliminer totalement les phases CALC (Calcul des scores), WTA (Mise à jour des états) et MAJ (Mise à jour de la liste des clones actifs) du décodage. Elle permet aussi d'éliminer le stockage des états des clones (Registre *Retats* dans les *PEs*). Cependant, elle nécessite de copier la liste temporaire vers la mémoire *Mca* à la fin du parcours de la matrice.

(6) Accès multiple à la matrice (ARCH06, Voir Annexe G) : Dans l'architecture de Dorrance et al. [92], la mémoire représente un goulet d'étranglement qui limite la performance de l'architecture. Si la plateforme offre l'accès parallèle à *Nm* mémoires, il est possible d'accélérer le parcours de la matrice en la répartissant sur ces différentes mémoires. Dans ce cas, l'architecture peut être étendue. Elle est maintenant composée d'un module central et de *Nm* modules périphériques, chacun accédant à une unique mémoire contenant une partie de la matrice. Chaque module périphérique est similaire à l'architecture proposée. La mémoire *MM* d'un module contient uniquement les zones : *lignes* et *asn*. Chaque mémoire identifiée par Id contient les lignes $ligW$ de la matrice telles que : $Id = ligW\ MOD\ Nm$ (Principe identique à l'allocation des lignes aux *PEs*). Le module central contrôle l'ensemble de l'architecture (calcul du score maximal du réseau, lecture des clones actifs, …). Chaque clone actif lu (resp. chaque colonne) est diffusé à tous les modules périphériques qui vont ensuite lire et traiter les clones associés (resp. lignes associées) à ce clone (resp. cette colonne). Les traitements sont donc réalisés en parallèle.

Le Tableau 6 montre pour chaque architecture l'apport en termes de gain en temps de calcul, ressources de mémorisation et ressources de calcul. Ces informations sont confirmées par les résultats présentés dans la section dédiée aux expériences.

Tableau 6 Récapitulatif des apports des différentes améliorations

Apport / Architectures	Gain en temps de calcul	Gain en coût de mémorisation requis	Gain en ressources de calcul
ARCH01 Fusion des phases du décodage	+	Non concerné	+
ARCH02 Stabilisation mémoire	+	+++	+
ARCH03 Allocation dynamique des zones	+	++	Non concerné
ARCH04 Réduction coût des comparateurs	Non concerné	Non concerné	++
ARCH05 Construction de la liste des clones actifs	++	-	Non concerné
ARCH06 Accès multiple à la matrice	+++	-	-

Le Tableau 7 résume les prérequis (couleur claire) et les compatibilités (couleur sombre) entre des différentes améliorations. Par exemple, ARCH05 nécessite l'amélioration introduite par ARCH04 (analyse le score des *PEs* au fur et à mesure, couleur claire). Elle est compatible uniquement avec ARCH06 (accès multiple à la matrice, couleur sombre).

Tableau 7 Prérequis et compatibilités entre les différentes améliorations

	ARCH01	ARCH02	ARCH03	ARCH04	ARCH05	ARCH06
ARCH01						
ARCH02						
ARCH03						
ARCH04						
ARCH05						
ARCH06						

Le Tableau 8 montre le temps de calcul des architectures après que les améliorations aient été progressivement introduites dans ARCH00.

Tableau 8 Temps de calcul des architectures après introduction progressive des différentes améliorations

Caractéristiques / Architectures	Temps de calcul (en cycles) $\boldsymbol{Tc}$
ARCH00 (Base)	$\frac{(\lvert Sccl\rvert+1)*Nt}{\lvert PE\rvert} + T_{CG} * Nlig * Nca + (\lvert Sccl\rvert + 7) * \frac{Nt}{\lvert PE\rvert} + 3 * Nt$
ARCH01 Fusion des phases de décodage	$\frac{(\lvert Sccl\rvert+1)*Nt}{\lvert PE\rvert} + T_{CG} * Nlig * Nca + 3 * Nt$
ARCH02 Stabilisation de la mémoire	$\frac{\boldsymbol{2*Nt}}{\lvert \boldsymbol{PE}\rvert} + T_{CG} * Nlig * Nca + 3 * Nt$
ARCH03 ($d \leq 30\%$) Allocation dynamique des zones	$\frac{\boldsymbol{Nt}}{\lvert \boldsymbol{PE}\rvert} + T_{CG} * Nlig * Nca + \boldsymbol{3 * Nt} + (\boldsymbol{d * Nt}) * \boldsymbol{\log_2(d * Nt)}$
ARCH04 Réduction coût des comparateurs	$\frac{Nt}{\lvert PE\rvert} + T_{CG} * Nlig * Nca + 3 * Nt + (d * Nt) * \log_2(d * Nt)$
ARCH05 Construction de la liste des clones actifs	$\frac{Nt}{\lvert PE\rvert} + T_{CG} * Nlig * Nca + (d * Nt) * \log_2(d * Nt)$
ARCH06 Accès multiple à la matrice (Nm accès)	$\frac{Nt}{\lvert PE\rvert} + \frac{\boldsymbol{T_{CG}*Nlig*Nca}}{\boldsymbol{Nm}} + (d * Nt) * \log_2(d * Nt)$

E. Evaluation de l'architecture générique et de ses améliorations

Afin de montrer l'effet des différentes améliorations, trois évaluations théoriques ont été réalisées pour différentes architectures :

- Temps de calcul ;
- Coût mémoire (en bits) pour le décodage d'un réseau précis ;
- Coût des ressources de calcul utilisées pour le décodage.

Chaque architecture évaluée intègre les améliorations précédentes. Par exemple, ARCH03 intègre les améliorations : ARCH00, ARCH01 et ARCH02.

Pour ces évaluations, la taille maximale des réseaux manipulables par les architectures doit être fixée. Ainsi, le réseau choisi est capable d'apprendre et de décoder des images de 28*28 pixels avec 256 niveaux de gris présentes dans la base de données d'images MNIST [105] (Mixed National Institute of Standards and Technology database). Pour manipuler ces images, un DNDM(1,784,256,256) contenant $28 * 28 = 784$ clusters (1 pixel = 1 cluster) et 256 neurones (1 neurone = 1 niveau de gris) est utilisé. Ce réseau contient en tout $Nt = 1 * 784 * 256 = 200.704$ clones. Dans les évaluations, toutes les mémoires ont une largeur standard de 32 bits ce qui est suffisant pour identifier ces 200.704 clones ($32\ bits \Leftrightarrow 2^{32}$ valeurs). L'image soumise au décodage est affectée par un bruit d'inversion qui modifie la couleur de quelques pixels. L'entrée contient alors 784 neurones (Un neurone étant associé à la couleur pour chaque position dans l'image).

Le nombre d'éléments de calcul de l'architecture est fixée à 64 (valeur utilisée dans [92]).

L'architecture ARCH06 n'est évaluée que dans la première comparaison car elle se focalise uniquement sur des accès multiples à la matrice, ce qui améliore les temps de calcul. Les évaluations ont été réalisées avec 8 accès multiples.

1. Comparaison du temps de calcul

Les temps de calcul des différentes architectures sont comparés dans cette section. Les différentes expressions de ces temps de calcul (Voir Tableau 8) montrent qu'ils dépendent de *Nca*, *Nlig, Nt* et $|PE|$. Ces paramètres peuvent avoir des valeurs indépendantes entre elles ce qui rend la comparaison difficile.

Afin de réaliser la comparaison des différentes architectures, Nt étant fixée, la valeur de $Nlig * Nca$ est ajustée en fonction de Nt.

Nlig est le nombre moyen de lignes contenant des éléments non nuls par colonne de la matrice. Ce nombre est équivalent au nombre moyen de poids synaptiques à 1 que possède un clone dans le réseau. Il est donc lié à la densité de la matrice. Pour que le réseau ait une bonne performance, il faut que la densité soit inférieure à 30% [62]. Ainsi, $Nlig \leq 30\% * Nt$.

Nca est le nombre de clones activés à l'initialisation du décodage. L'entrée est composée de 768 neurones actifs avec un neurone pour chaque cluster. Ces neurones (et leurs clones) représentent un pourcentage des neurones (des clones) du réseau. Dans ce réseau, en moyenne, un clone est alloué à chaque neurone. Ainsi, $Nca = 784$ et les clones activés représentent $\frac{784}{200.704} \approx 0{,}04\%$ des clones du réseau. Ainsi, $Nca = 0{,}04\% * Nt$.

La valeur $Nca * Nlig$ est alors bornée avec $0 < Nca * Nlig \leq 30\% * 0{,}4\% * Nt^2$.

Si d est la densité de la matrice, il est possible d'écrire $Nca * Nlig = d * 0{,}4\% * Nt^2$.

Dans cette évaluation, les ratio en temps de calcul sont donnés par rapport à la densité $d(W)$ qui varie entre 0,10% et 30%.

La Figure VI.8 montre le ratio en temps de calcul pour les différentes architectures et par rapport à ARCH00. Les résultats montrent que pour toutes les architectures, ce ratio diminue en fonction de la densité. Il se stabilise à partir de 3% de densité pour toutes les architectures. Cela signifie plus la densité baisse, plus le gain est élevé. La baisse et la stabilisation du gain est due à l'expression $Nlig * Nca$ qui s'impose face aux autres paramètres. Cela démontre la dépendance de la performance de l'architecture par rapport à la lecture de la mémoire.

Pour des densités inférieures à 3%, ARCH05 divise au minimum le temps de calcul par 2. Ainsi, la création dynamique de la liste des clones actifs apporte un avantage conséquent car elle élimine des phases entières du décodage. L'architecture ARCH06 divise au maximum le temps par 15 et au minimum par 8 ce qui montre l'intérêt d'un accès parallèle à plusieurs mémoires.

D'un autre côté, toutes les améliorations proposées ne dégradent pas le temps de calcul par rapport à l'architecture de base.

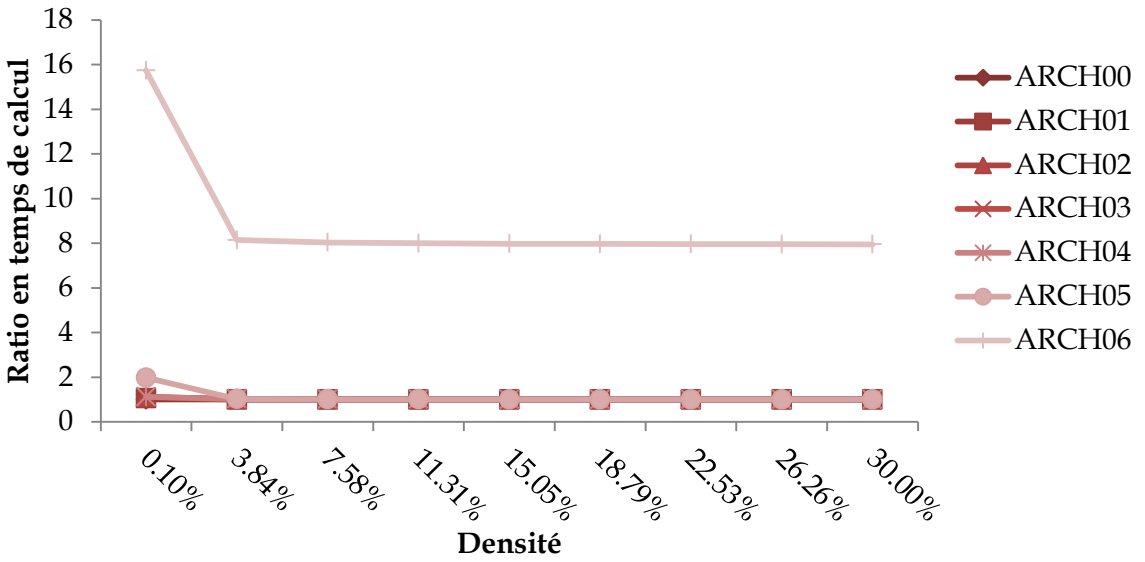


Figure VI.8 Ratio en temps de calcul des améliorations par rapport à ARCH00 et en fonction de la densité.

2. Comparaison du coût de mémorisation requis pour le décodage du réseau

La Figure VI.9 montre les coûts de mémorisation requis pour réaliser le décodage du réseau.

Concernant les architectures utilisant l'allocation dynamique (ARCH03 à ARCH05), le pourcentage maximal de clones concerné par le décodage est fixé à 50% par élément de calcul (*PE*). Cela signifie chaque *PE* peut manipuler au maximum 50% des clones auquel il

est associé. Cette valeur a été choisie volontairement supérieure aux densités réelles (≤ 30%) afin d'atténuer le manque d'espace lors de l'allocation dynamique.

Les résultats nous montrent que les architectures ARCH00 et ARCH01 nécessitent près de 10 fois plus de mémoire par rapport aux autres architectures. Ce résultat est dû à la profondeur conséquente des zones logiques ($\left\lceil \frac{768}{32bits} \right\rceil = 24$). Cela montre que la stabilisation de la profondeur des zones logiques introduite dans ARCH02 a un impact important sur les possibilités de l'architecture. L'allocation dynamique introduite dans ARCH03 permet d'obtenir un gain de 25% par rapport à ARCH02. D'un autre côté, l'augmentation du coût mémoire due à l'introduction de la mémoire temporaire des clones actifs dans ARCH05 est faible (+14%).

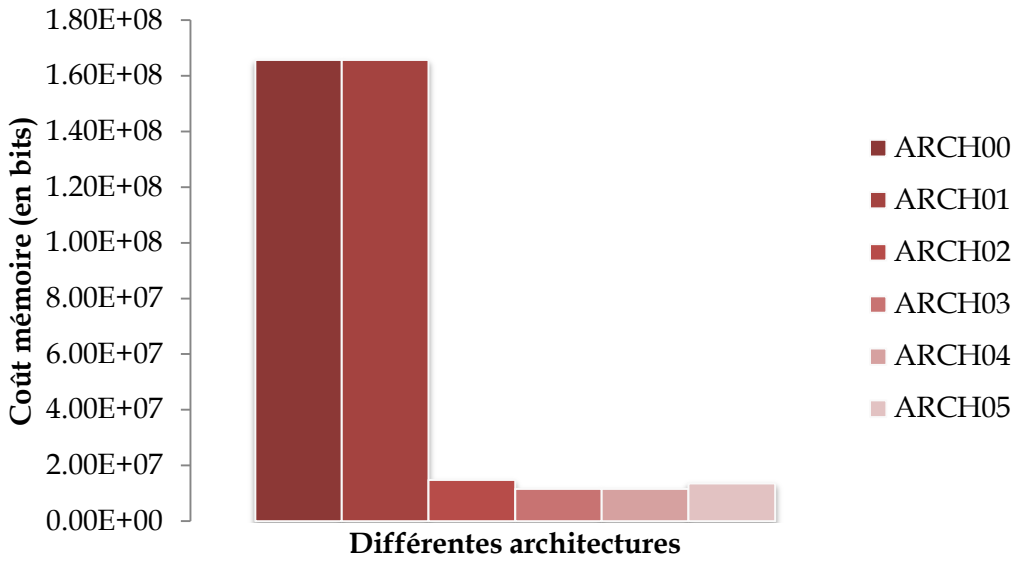


Figure VI.9 Nombre de bits requis pour réaliser le décodage d'un réseau DNDM(1,784,256,256)

3. Comparaison du coût des ressources de calcul requis pour le décodage

La Figure VI.10 montre le coût en Portes NAND Equivalentes (PNE) des ressources de calcul uniquement requis par le décodage (les ressources requises, le routage des données, pour le contrôle et pour les accès mémoires ne pris en compte). Ce coût est évalué en fonction de la largeur des mots des mémoires des architectures. Les largeurs choisies sont de 16, 32 et 64 bits. Les coûts additionnels ajoutés par l'architecture ARCH04 et ARCH05 (multiplexeurs permettant d'obtenir les scores de chaque *PE)* ont aussi été inclus dans les évaluations.

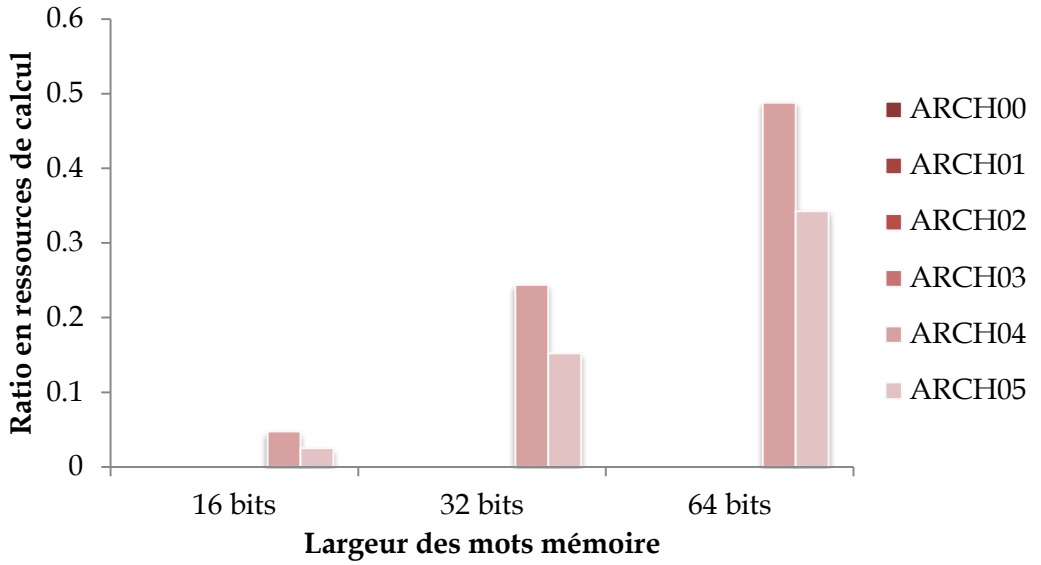


Figure VI.10 Ratio en ressources de calcul par rapport à ARCH00 et en fonction des largeurs des mots mémoires

Les architectures ARCH01, ARCH02, ARCH03 apportent un gain variant entre 3% (32 bits) et 9% (64 bits) par rapport à ARCH00. Les architectures ARCH04 et ARCH05 apporte un gain plus important (27% à 32%) par rapport à ARCH00. Cela montre que l'arbre de comparateurs présent dans le contrôleur de ARCH00 représente un module de calcul ayant un impact important.

F. Bilan

Dans ce chapitre, un nouveau mode de stockage des matrices parcimonieuses et une première architecture générique permettant de mettre en œuvre des réseaux à clones ont été présentés.

Le mode de stockage proposé s'inspire du mode CSC et consiste à réduire le nombre de bits de codage des positions des éléments non nuls au sein de la matrice. Les résultats des évaluations nous montrent qu'il permet d'obtenir des gains en coût de stockage pour une augmentation faible de la complexité temporelle.

L'architecture proposée se base sur une architecture performante réalisant des multiplications de matrices parcimonieuses par des vecteurs (SpMxVs). De cette architecture, elle conserve la structure globale et la méthode de routage des informations tout en intégrant de nombreuses modifications afin de mettre en œuvre toutes les opérations réalisées dans les réseaux à clones. Parmi ces modifications figurent l'intégration d'un système de score des clones ainsi que le calcul du score maximal du réseau. Des améliorations ont été proposées afin de réduire le temps de calcul, le coût mémoire et le coût

en ressources de calcul. Cette architecture et ses améliorations ont été évaluées en termes de temps de calcul, coût de mémorisation et de coût en ressources de calcul. Les résultats de ces évaluations montrent que la meilleure architecture est celle qui utilise une allocation dynamique des zones logiques des clones ainsi qu'une construction dynamique de la liste des clones actifs (ARCH05). Si la plateforme visée possède plusieurs mémoires pouvant contenir une partie de la matrice, un accès multiple à la matrice peut être réalisé. En ce sens, une architecture (ARCH06) ayant cette capacité a aussi été présentée et son effet sur le temps de calcul a été évalué.

CHAPITRE VII. Conclusion Générale et Perspectives

Conclusion générale

Les objectifs de cette thèse étaient les suivants : (1) concevoir des modèles inspirés du modèle GBNN et plus performants que l'état de l'art, (2) proposer des architectures moins coûteuses que les solutions existantes et (3) concevoir une architecture générique configurable mettant en œuvre les modèles proposés et capable de manipuler des réseaux de tailles variables.

Concernant les modèles améliorant les performances, le concept de *clones* et de *réseaux à clones* ont été présentés. Il existe deux classes de réseaux à clones : les réseaux à allocation statique de clones où le nombre de clones alloués à chaque neurone est constant et les réseaux à allocation dynamique de clones où le nombre de clones alloué à chaque neurone peut augmenter. Les différentes variantes de réseaux à clones ont été présentées selon ces deux classes. Certaines d'entre elles généralisent les approches de l'état de l'art. Toutes ces variantes ont des structures et des algorithmes d'apprentissage différents mais elles possèdent toutes la même dynamique de décodage. Elles conservent aussi les mêmes bases (clusters, poids binaires, calculs simples) et les mêmes fonctionnalités (restitution de messages) que le modèle GBNN originel. Les résultats des évaluations de ces modèles montrent que les meilleurs modèles surpassent l'état de l'art. Les caractéristiques de ces modèles ayant contribuées à leur performance sont : l'usage d'un algorithme compétitif, la présence de plusieurs sous-réseaux et l'usage d'une allocation dynamique de clones aux neurones.

Par rapport au second objectif, une simplification du modèle originel et des architectures matérielles y relatives ont été présentées. La simplification remplace les calculs entiers par des calculs logiques, ce qui réduit la complexité des calculs. Les différentes architectures basées sur cette simplification introduisent deux améliorations. La première prend en compte la symétrie de la matrice d'adjacence dans le modèle GBNN et permet de diviser les coûts des ressources requises pour l'apprentissage par 2. La seconde introduit la sérialisation des communications et des calculs ce qui permet de réduire les coûts en termes de ressources malgré un temps de calcul plus long mais constant. Ces différentes architectures ont été évaluées en termes de coût et de fréquence. Les résultats montrent que les architectures proposées réduisent le coût en ressources matérielles par rapport à l'état de l'art tout en ayant une fréquence de fonctionnement plus grande.

Concernant le troisième objectif, une architecture générique et des améliorations permettant de mettre en œuvre des réseaux à clones ont été proposées. L'architecture proposée s'inspire d'une architecture performante mettant en œuvre des multiplications de matrices parcimonieuses par des vecteurs. Celle-ci inclut les opérations spécifiques aux réseaux à clones (Calcul du score maximal, manipulation de la valeur des clones, …). Les améliorations consistent principalement à fusionner les phases de calcul, à stabiliser l'espace mémoire alloué à chaque clone et à allouer la mémoire uniquement aux clones concernés par le décodage. L'architecture et l'effet des différentes améliorations ont été évalués. Les résultats montrent que les améliorations permettent de réduire le temps de calcul requis pour une itération de décodage, le coût de mémorisation requis pour le décodage, ainsi que le coût en ressources de calcul.

Perspectives

Modèles et réseaux à clones

Des modèles et des algorithmes peu complexes ont été proposés pour les réseaux à clones. Parmi ces modèles, certaines surpassent l'état de l'art mais sont paramétrés arbitrairement. Une méthode pour déterminer le paramètre automatiquement reste à définir.

Les résultats ont montré que les caractéristiques de ces modèles ayant contribué à leur bonne performance sont : la présence de plusieurs sous-réseaux, l'usage d'un algorithme compétitif et l'allocation dynamique de clones. Aucun des modèles présentés n'intègre toutes ces caractéristiques. Un tel modèle reste à concevoir et à évaluer.

Des modèles similaires aux réseaux à clones comme les réseaux basés sur le Cogent-Confabulation peuvent bénéficier de l'usage de clones. En effet, ces réseaux y sont proches en termes de structure (usage de clusters) et de dynamique (calcul de score et Winner Take All). Leur performance dépend aussi de la densité de leurs connexions. L'intérêt d'une telle étude est dû à l'utilisation de ces réseaux pour des applications de classification et pour la mise en œuvre des systèmes d'intelligence artificielle.

Architectures matérielles génériques

Une architecture générique et des améliorations ont été proposées pour mettre en œuvre des réseaux à clones de différentes tailles. Cependant, elles n'ont pas été réalisées. Il est nécessaire de les mettre en œuvre et de les synthétiser afin de les évaluer en termes de coût en ressources de calcul, de fréquence maximale ainsi qu'en termes de nombre de cycles de décodage pour des applications réelles (Par exemple : traitement d'image).

Le décodage dans les réseaux à clones s'apparente à la multiplication d'une matrice parcimonieuse par un vecteur. De nombreuses mises en œuvre de ce type de calcul existent

pour différentes plateformes autre les FPGA (CPU, GPU, …). L'introduction du principe de calcul proposée dans ce document dans d'autres plateformes peut être étudiée et les résultats peuvent être analysés.

Un mode de stockage simple et performant des matrices parcimonieuses a été proposé. Cependant, d'autres modes de stockage possédant des gains en coût de mémorisation, des comportements et des complexités différents existent dans l'état de l'art. Une comparaison de cette proposition à ces modes reste à réaliser.

Annexe

A. Mise à jour de la contribution d'un cluster pour un clone

Afin de mettre à jour la contribution d'un cluster distant pour un clone, il faut connaître l'adresse du bit qu'il faut mettre à 1. Pour un cluster distant $\mathcal{C}_{(j',k)}$, cette adresse est composée de deux sous-adresses $AdrM$ et $AdrB$ qui sont données par la formule suivante :

$$j' = AdrM * La(Mint) + AdrB \quad (66)$$

Avec $AdrM$ étant l'adresse relative du mot dans la partie $Sccl$ de la zone logique du clone où le bit doit être modifié et $AdrB$ le numéro du bit qui doit être modifié dans ce mot.

La Figure 0.1 montre l'aperçu de la mémoire interne d'un élément de calcul (*PE)* ainsi que les sous-adresses permettant de mettre à jour la contribution d'un cluster pour un clone.

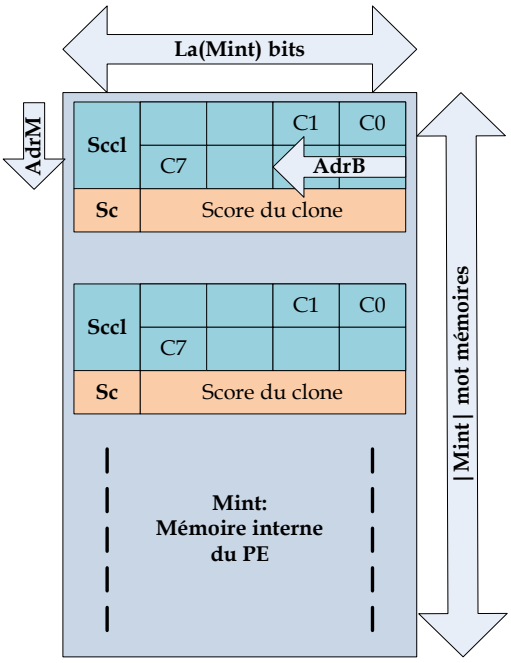


Figure 0.1 Données permettant la mise à jour la contribution d'un cluster pour un clone

Afin de déterminer où mettre à jour la zone logique associée à un clone identifié par $Idcl$, trois informations doivent être déterminées à travers deux calculs. Le premier calcul permet d'obtenir $\#PE$ qui est le numéro du PE auquel est associé le clone et $NumZ$ le numéro de la zone dans la mémoire interne de ce PE à laquelle est associé ce clone. $Idcl$ est donné par la formule suivante :

$$Idcl = |PE| * NumZ + \#PE \quad (67)$$

Ce calcul est une division entière.

Une autre méthode pour obtenir $\#PE$ et $NumZ$ à partir de $Idcl$ est basée sur le codage binaire des valeurs. $Nbc(val)$ étant le nombre de bits permettant de coder la valeur val, si $Nbc(Idcl) = nb1$ et que $Nbc(\#PE) = nb2 < nb1$, alors $\#PE$ est obtenu en sélectionnant les $nb2$ bits de poids faibles de $Idcl$. Dans ce cas, $NumZ$ est donné par les $nb1 - nb2$ bits de poids forts.

Exemple : En utilisant le première solution, si $Idcl = 100$ et $|PE| = 64$, $Idcl = 64 * NumZ + \#PE$. Ce qui donne $\#PE = 36$ et $NumZ = 1$.

En utilisant la seconde solution, si les identifiants des clones sont codés sur $nb1 = 16bits$, alors $IdCl = 100_{10} = (0000\_0000\_0110\_0100)$. $nb2 = 6bits$ car $2^6 = 64PEs$. Ainsi, $\#PE = (10\_0100) = 36$ tandis que $NumZ = (0000\_0000\_01) = 1$.

Le second calcul permet de déterminer l'adresse $Adr(Idcl)$ de la zone $NumZ$ dans la mémoire interne *Mint*. Cette adresse est obtenue en multipliant le numéro de la zone par la profondeur de la zone :

$$Adr(Idcl) = NumZ * (|Sccl| + 1) \quad (68)$$

En utilisant les données précédentes, $Adr(Idcl) = 1 * (1 + 1) = 2$.

B. Fusion des phases de décodage (ARCH01)

Dans un objectif de réduire la latence globale du décodage, il est possible de fusionner les phases de décodage en réalisant l'étape du calcul du score des clones CALC en même temps que l'étape de détermination des contributions des clusters PREC. Le principe de cette fusion est le suivant : lorsqu'une zone logique dans un élément de calcul *PE* est mise à jour, le score du clone associé est directement calculé. Cela permet de mettre directement à jour le score maximal du *PE* dans *Rlmax* ainsi que le score maximal du réseau. Si jamais le nouveau score du clone est supérieur au contenu de *Rlmax,* alors le contenu de *Rlmax* est remplacé par ce score.

En effet, lors de la phase PREC, si un *PE* est sollicité par le contrôleur *CG* pour mettre à jour la zone logique d'un clone, il existe un délai avant que ce *PE* soit de nouveau sollicité par *CG*. Il est alors possible d'utiliser ce délai pour réaliser d'autres tâches.

1. Effet sur le temps de calcul

A la fin de la phase PREC, le score de tous les clones et le score maximal du réseau sont disponibles. La phase de calcul de score CALC a disparu. Le temps de calcul devient :

$$Tc = \frac{(|Sccl| + 1) * Nt}{|PE|} + T_{CG} * Nlig * Nca + 3 * \frac{Nt}{|PE|} + 3 * Nt\ cycles \quad (69)$$

On constate que le temps de calcul (69) est maintenant indépendant de la profondeur de la zone logique des clones et qu'il est inférieur au temps de calcul originel (65).

2. Effet sur les ressources de calcul

Cette optimisation permet aussi de simplifier le module *Tadd* (arbre d'additionneurs) des éléments de calcul PEs qui était utilisé pour calculer le score des clones. En effet, lorsque que la zone associée à un clone doit être mise à jour (changement d'un unique bit) et si le calcul du score est réalisé en même temps, le score de ce clone est incrémenté si et seulement si le bit modifié passe effectivement de 0 à 1. Cela revient à prendre directement en compte la contribution d'un cluster distant. Ainsi, l'arbre d'additionneurs n'est plus nécessaire et peut être remplacé par un *additionneur simplifié.*

La Figure 0.2 montre un aperçu d'un tel additionneur. Celui-ci prend en entrée d'une part le score actuel du clone et d'autre part, l'inverse de la valeur du bit à modifier. L'optimisation permet donc d'obtenir un gain en ressources de calcul.

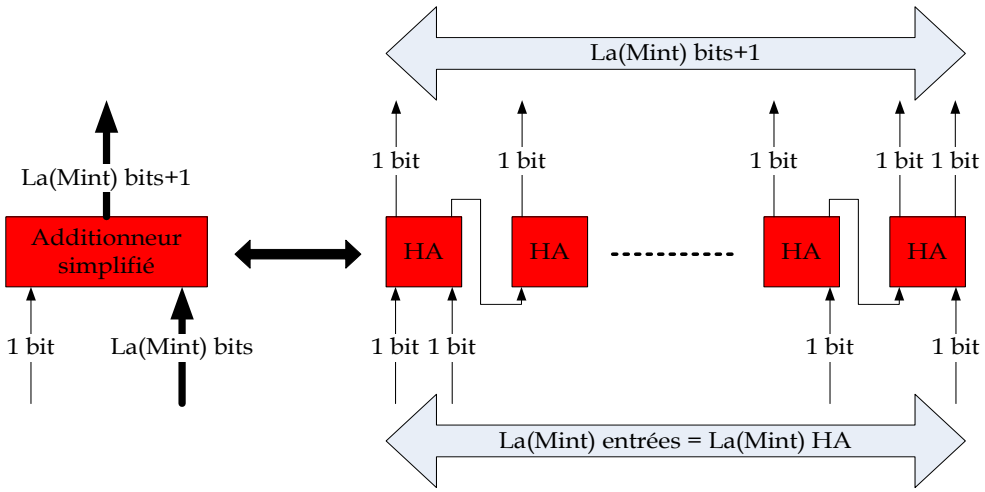


Figure 0.2 Schéma de l'additionneur simplifié

HA (Half-Adder) représente un demi-additionneur.

Dans la suite, l'évaluation du gain en ressources est réalisée en Portes NAND Equivalentes (PNE).

Si C_{HA} est le coût d'un demi-additionneur, et si $C_{FA}(i)$est le coût d'un additionneur complet de i bits alors un arbre d'additionneur avec $nbits$ ports de 1 bit possède un coût de :

$$Tadd(nbits) = \frac{nbits}{2} C_{HA} + \sum_{etg=1}^{\log_2(nbits)} (\frac{nbits}{2^{i+1}}) C_{FA}(i+1) \quad (70)$$

En effet, un arbre d'additionneurs contient un étage de $\frac{nbits}{2}$ demi-additionneurs suivi de plusieurs étages d'additionneurs. La taille des entrées des additionneurs augmente de manière incrémentale en fonction de l'étage. Par exemple, à l'étage 0, l'entrée des additionneurs est de 1 bit. A l'étage 1, l'entrée sera de 2 bits, etc. Le nombre d'additionneurs d'un étage est divisé par 2 par rapport à l'étage inférieur.

La coût minimal d'un arbre d'additionneurs peut être estimée en remplaçant $C_{FA}(i+1)$ par $C_{FA}(2)$. En effet, $C_{FA}(i+1) \geq C_{FA}(2), \forall i \geq 1$, c.à.d. le coût minimal d'un additionneur est supérieur à celui d'un additionneur dont chaque entrée est sur 2 bits.

$$Tadd(nbits) = \frac{nbits}{2} C_{HA} + \sum_{etg=1}^{\log_2(nbits)} (\frac{nbits}{2^{i+1}}) C_{FA}(2) \quad (71)$$

Sachant que $C_{FA}(2) = 2 * C_{HA}$, on obtient :

$$\begin{aligned} Tadd(nbits) &= \frac{nbits}{2} C_{HA} + 2 * C_{HA} * nbits * \sum_{etg=1}^{\log_2(nbits)} \left(\frac{1}{2^{i+1}}\right) \\ &= C_{HA} * nbits * \left(\frac{1}{2} + 2 * \sum_{etg=1}^{\log_2(nbits)} \left(\frac{1}{2^{i+1}}\right)\right) \\ &= C_{HA} * nbits * \left(\frac{1}{2} + \frac{2}{4} * \frac{1 - \frac{1}{2}^{\log_2(nbits)}}{1 - \frac{1}{2}}\right) \\ &= C_{HA} * nbits * \left(\frac{1}{2} + 1 - \frac{1}{2}^{\log_2(nbits)}\right) \\ &= C_{HA} * nbits * \left(\frac{3}{2} - \frac{1}{2}^{\log_2(nbits)}\right) \simeq C_{HA} * nbits * \left(\frac{3}{2}\right) \end{aligned} \quad (72)$$

Or l'additionneur simplifié qui remplace l'arbre d'additionneurs ne contient que des demi-additionneurs et possède un coût de :

$$Adds = C_{HA} * nbits \quad (73)$$

Le gain en ressource de calcul par rapport à l'arbre d'additionneur est alors (au minimum) de $1 - \frac{Tadds(nbits)}{Adds(nbits)} = 1 - \frac{3}{2} = 50\%$.

C. Stabilisation de la profondeur des zones logiques (ARCH02)

Dans l'architecture proposée, les zones logiques dans les mémoires internes *Mint* des *PEs* ont un coût de mémorisation qui dépend de $|Sccl|$. $|Sccl|$ varie en fonction du nombre de clusters par sous-réseau C. Il est possible de rendre la profondeur des zones logiques constante et indépendante de *C.* Ainsi, cela permet qu'avec la même configuration de la mémoire qu'une architecture ARCH00 ou ARCH01, il soit possible manipuler des réseaux de plus grandes tailles.

En effet, le parcours de la liste des clones actifs s'assimile au parcours de leur cluster associé. Pour un clone actif lu, les contributions de son cluster pour les clones connectés à ce clone doivent être mises à jour. Si la liste des clones actifs est triée selon leur cluster associé, alors le calcul des scores revient à considérer progressivement les contributions provenant de chaque cluster distant. Dans ce cas, le score d'un clone *Idcl* est incrémenté de 1 si le cluster en cours contient au moins un clone actif ayant un poids à 1 avec *Idcl* et si ce cluster est différent du dernier cluster qui a mis à jour ce clone.

Au sein de l'architecture, cela se traduit par deux modifications :

- Une modification du format des zones logiques car les parties *Sccl* requises pour stocker les contributions disparaissent. Dans le nouveau format des zones logiques, la partie $Sccl$ est remplacée par un unique mot mémoire $LastIdcl$. Celui-ci contient l'identifiant du cluster associé au dernier clone actif ayant provoqué la mise à jour du score de ce clone.
- Une modification du processus de mise à jour du score des clones. Pour un élément de calcul, en plus de recevoir l'adresse du clone qui doit être modifié, celui-ci doit aussi recevoir l'identifiant du cluster auquel appartient le clone actif en cours. Cette information permet de le comparer à $LastIdCl$. Si, les deux sont différents, le nouvel identifiant du cluster doit remplacer l'ancien et le score du clone doit être incrémenté de 1.

$Idcl$ étant le clone actif en cours, la mise à jour du score d'un clone se déroule 3 étapes :

- Lecture de la valeur contenue dans $LastIdcl$.
- Comparaison du cluster associé au clone $Idcl$ à $LastIdcl$ pouvant aboutir à deux résultats possibles :

- Si les valeurs sont égales, alors les clusters sont identiques. Dans ce cas, le score n'est pas incrémenté.
- Si les valeurs ne sont pas égales, alors les clusters sont différents. Dans ce cas, la valeur contenue dans $LastIdcl$ est remplacée par $Idcl$ tandis que le score du clone est incrémenté de 1.

En utilisant cette amélioration, la latence maximale de calcul de chaque *PE* est de 5 cycles : (1) lecture de $LastIdCl$, (2) comparaison à $IdCl$, (3) incrémentation du score si différent, (4) comparaison à $Rlmax$/Ecriture du score dans la mémoire; (5) mise à jour de $Rlmax$.

La modification proposée nécessite que la liste des clones actifs soit ordonnée par cluster (indépendamment de l'ordre des clusters selon leur identifiant). Cela n'est possible que si :

- La liste des clones actifs est ordonnée (par cluster) au début de chaque itération de décodage.
- La liste des clones actifs est construite en étant ordonnée (à la phase MAJ de mise à jour de la liste des clones actifs).

Le processus originel utilisé dans la phase MAJ (Voir CHAPITRE VI.C.7.e)) assure que la liste des clones actifs soit classée selon leur cluster. En effet, lors de cette phase, le contrôleur *CG* parcourt les *PEs* de manière consécutives. Or, pour deux clones d'identifiants consécutifs $Idcl1$ et $Idcl2$ avec $Idcl2 = Idcl1 + 1$, ceux-ci sont associés à des *PEs* consécutifs. Etant donné que le parcours des *PEs* pour la création de la liste des clones actifs suit le même ordre, alors $Idcl1$ est toujours inférieur à $Idcl2$. Ainsi, le cluster auquel appartient $IdCl1$ a toujours un identifiant inférieur à celui auquel appartient $Idcl2$.

Un autre avantage de cette approche est d'éliminer l'opération de calcul des adresses où la contribution doit être mise à jour (Voir CHAPITRE VII.A). En effet, la nouvelle approche n'a plus besoin de calculer l'adresse du bit à mettre à jour. Cela apporte donc un gain en ressource de calcul.

La Condition de Décodage sans Stockage Externe CDSE est aussi modifiée car les zones logiques ont désormais une profondeur constante de 3 mots mémoire. Ainsi, la condition devient :

$$\begin{gathered} Mtsc = 2 * Nt \text{ mots mémoire} \\ Mtsc \leq (2) * |PE| * \left\lfloor \frac{|Mint|}{(2)} \right\rfloor \text{ mots mémoire} \end{gathered} \tag{74}$$

Cette condition est nettement meilleure que celle offerte par l'architecture originelle ARCH00 à travers l'expression (59) car elle ne dépend plus la profondeur des zones logiques.

Le temps de calcul global pour une itération de décodage est alors égal à :

$$Tc = \frac{2*Nt}{|PE|} + T_{CG} * Nlig * Nca + 3 * Nt \text{ cycles} \quad (75)$$

D. Allocation dynamique des zones logiques (ARCH03)

Dans l'architecture ARCH00, les mémoires internes *Mint* des *PEs* sont segmentées en zones logiques. Chaque zone logique est allouée à un clone précis et ceci pour chaque clone du réseau. Cette organisation statique a été utilisée car elle s'inspire de l'organisation originelle de la mémoire existante dans l'architecture de Dorrance et al. (Voir CHAPITRE IV.E) (Allocation d'un mot mémoire pour chaque ligne du vecteur de sortie équivalent à l'allocation d'une zone pour chaque clone du réseau). Cependant, lors du décodage, certaines zones logiques ne sont pas manipulées car leur clone associé n'est pas concerné par ce décodage.

La probabilité qu'un clone soit concerné par un décodage dépend de la densité globale du réseau (aussi égale à la moyenne des densités locales pour tous les clones du réseau). Par exemple, si la densité globale est de 10%, alors seules 10% des zones logiques seront mis à jour lors du décodage ce qui rend 90% des zones logiques inutiles.

Il se pose donc ici un problème d'allocation des zones logiques, une allocation qui est statique dans les architectures ARCH00, ARCH01 et ARCH02.

Afin de résoudre ce problème, une allocation dynamique des zones logiques aux clones peut être réalisée. Pour mettre en œuvre cette fonctionnalité, l'architecture de base doit être modifiée. Pour illustrer ces modifications, nous nous baserons sur l'architecture ARCH02.

1. Modification de l'architecture

La première modification consiste à ajouter une mémoire supplémentaire locale à chaque *PE*. En effet, étant donné que les adresses où se trouvent les zones logiques des clones ne sont plus statiques, il est nécessaire d'utiliser une mémoire *Madrcl* pour stocker ces adresses. Cette mémoire doit être initialisée au début de chaque itération de décodage. Elle est modifiée au fur et à mesure de l'allocation. Sa profondeur $|Madrcl|$ est égale au nombre maximal de clones associé au *PE* et sa largeur $La(Madrcl)$ est égale au nombre de bit requis pour pointer n'importe quelle ligne de la matrice interne *Mint* d'un *PE*, c.à.d. $La(Madrcl) = Nbc(|Mint| + 1)$. Une valeur supplémentaire est ajoutée afin de coder la valeur initiale. Par exemple, si $|Mint| = 512$ mots mémoires, alors $La(Madrcl) = 9 + 1 = 10$ bits.

L'usage de l'allocation dynamique élimine aussi la nécessité d'initialiser toute la mémoire *Mint* des *PEs*. En effet, l'initialisation s'effectue uniquement lorsqu'une allocation est requise

et ne s'effectue que pour la zone logique allouée. Cependant, toute la mémoire $Madrcl$ qui contient pour chaque clone l'adresse de sa zone logique doit être initialisée.

Une seconde modification concerne les zones logiques dans les mémoires *Mint*. Celles-ci sont désormais constituées de trois mots mémoire :

- Le premier mot contient l'identifiant du clone associé à cette zone ;
- Le second mot contient $LastIdcl$ qui contient l'identifiant du dernier cluster qui a mis à jour ce clone.
- Le troisième mot Sc qui contient le score du clone.

2. Modification du fonctionnement du décodage

En plus de l'architecture, trois phases du décodage doivent être modifiées :

- INIT : car les mémoires *Madrcl* doivent être initialisées à chaque itération du décodage.
- PREC : car l'allocation dynamique doit pouvoir allouer des zones logiques aux clones connectés aux clones actifs.
- MAJ : car la liste des clones actifs n'est plus ordonnée par cluster après les avoir récupérés des éléments de calcul. En effet, le parcours des PEs ne concernera que les clones concernés par le décodage. Dans ce cas, après avoir stocké les clones actifs dans la mémoire contenant les clones actifs *Mca*, une étape supplémentaire pour trier la liste des clones actifs par cluster grâce un algorithme de tri (quick-sort, intro-sort, heap-sort, etc.)

Pour la phase PREC, lorsqu'un *PE* reçoit des informations afin de mettre à jour le score d'un clone $Idcl$, les étapes suivantes sont réalisées :

- Lecture de la mémoire *Madrcl* pour vérifier si une zone a déjà été allouée à ce clone
- Si l'allocation est requise, vérification de l'existence de zones logiques disponibles pour réaliser une allocation, puis allocation en deux temps : (1) inscription de l'adresse de cette zone dans *Mardcl* et (2) initialisation cette zone (identifiant du clone associé, identifiant du cluster, score à 1).
- Si l'allocation n'est pas requise, traitement normal de la zone allouée au clone (comparaison du cluster distant à *LastIdcl*, mise à jour du score du clone si nécessaire, …).

La latence du *PE* est alors augmentée de 2 cycles et passe de 4 à 6 cycles à cause de l'ajout du test de l'allocation et des étapes d'initialisation.

L'usage de l'allocation dynamique modifie la Condition de Décodage sans Stockage Externe (CDSE) qui détermine si un décodage peut être réalisé pour un réseau en particulier. Pour une architecture avec allocation dynamique, elle est donnée par :

$$Mtsc \leq (3) * \tau * |PE| * \left\lfloor \frac{|Mint|}{(3)} \right\rfloor \text{ mots mémoire} \quad (76)$$

Avec $\tau < 1$. τ représente le pourcentage maximal de clones associés à un PE et qui pourront bénéficier d'une zone logique.

Concernant temps de calcul global, si Nca clones actifs sont lus lors du décodage, et si chaque clone actif nécessite de considérer $Nlig$ clones, alors le nombre total de clones manipulés par les *PEs* lors de la phase de mise à jour de la liste des clones actifs (MAJ) est $b * Nlig * Nca$ $(b < 1)$. Cette valeur est toujours inférieure au nombre total de clones du réseau Nt. En effet, le nombre maximal de clones actifs est toujours inférieur au nombre de clones du réseau.

Le temps de calcul est aussi affecté si l'on inclut les optimisations ARCH01 permettant de stabiliser les profondeurs des zones logiques. En effet, afin bénéficier de cette optimisation, il est nécessaire de trier la liste des clones actifs au début ou à la fin de chaque itération de décodage. Le nombre de clones actifs à la fin d'une itération de décodage n'est pas forcément identique au nombre présent au début de cette même itération de décodage. Au maximum, il est égal au nombre maximal de clones distants que peut élire un clone par le WTA, c.à.d. $Nlig = d * Nt$ (Nombre de connexions en moyenne mis à 1 dans le réseau).

Les meilleurs algorithmes de tri ont une complexité de l'ordre de $n * \log(n)$ [106] (Avec n le nombre d'éléments). Ainsi le temps de calcul devient :

$$Tc = \frac{Nt}{|PE|} + T_{CG} * Nlig * Nca + 3 * Nt + (d * Nt) * \log_2(d * Nt) \text{ cycles} \quad (77)$$

E. Elimination de l'arbre de comparateurs dans le contrôleur (ARCH04)

Bien que le module *MIN-MAX* dans le contrôleur *CG* ne soit pas le plus grand consommateur en termes de ressources, il est possible de le simplifier et de réduire son coût.

Lorsque le score d'un clone est mis à jour dans un *PE*, le score maximal du *PE* contenu dans $Rlmax$ est aussi mis à jour. Cela implique d'une part que ce *PE* est en avance par rapport à d'autres *PEs*. Cette avance accorde un délai qui peut permettre au contrôleur de pouvoir interagir avec la valeur fournie par le *Rlmax* de ce *PE* sans influencer le décodage.

Afin de réaliser cette tâche, le module *MIN-MAX* est modifié. L'arbre des comparateurs est remplacé par une mémoire de taille fixe *Midpe* (Mémoire des identifiants des *PEs*) et un timer. Lorsqu'un clone d'un *PE* doit être mis à jour, le contrôleur stocke l'identifiant de ce *PE* dans cette mémoire. Ensuite, le module attend quelques cycles (Attente du timer) afin de donner le temps à ce *PE* de mettre son maximum à jour. Quand ce délai arrive à sa fin, le module compare le score maximal de ce *PE* au score maximal du réseau avant de les mettre à jour si nécessaire. Pendant ce temps, d'autres *PEs* peuvent être ajoutés à la liste des *PEs* à traiter.

Ainsi, l'arbre de comparateurs composé de $|PE| - 1$ éléments est remplacé par un unique comparateur. Cette amélioration augmente légèrement le coût en ressources de stockage de l'architecture à cause de l'ajout de la mémoire *Midpe*. Cependant, ce coût est constant quel que soit la configuration. En effet, sa profondeur est égale au nombre minimal de cycles séparant la mise à jour du score d'un clone par deux *PEs*. Elle est donc égale à 6 qui est la latence d'un *PE*.

F. Construction dynamique de la liste des clones actifs (ARCH05)

L'objectif de cette partie est de réduire le temps de calcul global en construisant la liste des clones actifs directement lors de la phase de calcul de score PREC. Pour cela, une mémoire *Mca2* contenant la liste temporaire des clones actifs de la prochaine itération est utilisée. Celle-ci est modifiée à chaque fois que le score maximal du réseau est mis à jour. Une autre mémoire est aussi nécessaire pour stocker les identifiants des clones dont la zone logique doit être mise à jour. A chaque mot de cette mémoire est associé un mot de la mémoire *Midpe* qui stocke les numéros des éléments de calcul (*PEs*) associés à ces clones.

Lorsqu'un élément de calcul (*PE)* doit mettre à jour le score d'un clone, celui-ci provoque la mise à jour du score maximal du réseau. Ainsi, l'architecture dispose toujours d'un score maximal.

Si le score maximal du réseau est modifié et étant donné que le clone responsable de cette mise à jour est connu, alors la liste temporaire des clones actifs (*Mca2*) doit être réinitialisée et ce clone doit être ajouté à cette liste. En effet, celui-ci possède le score maximal du réseau tandis que les clones de la liste possèdent un score inférieur. Si le score de ce clone a effectivement changé et si ce clone possède un score égal au score maximal du réseau, alors il (son identifiant) doit être ajouté à cette liste. Si son score est inférieur, alors ce clone est ignoré.

L'avantage de créer une liste de clones actifs durant le calcul de score permet d'éliminer explicitement les phases CALC (calcul de score), WTA (calcul du score maximal et mise à jour des états des clones) et MAJ (Mise à jour de la liste des clones actifs).

Le nombre maximal d'éléments que peut contenir *Mca2* à la fin du parcours de la matrice est égal au nombre d'éléments lié à une connexion $d * Nt$.

Ainsi, le temps de calcul global dépend du temps nécessaire pour parcourir la matrice ainsi que temps nécessaire pour trier la mémoire *Mca2* et copier son contenu:

$$Tc = \frac{Nt}{|PE|} + T_{CG} * Nlig * Nca + d * Nt + (d * Nt) * \log_2(d * Nt) \text{ cycles} \quad (78)$$

Ce temps est inférieur à celui obtenu avec les architectures précédentes. Il faut cependant noter que l'usage d'une mémoire pour stocker une liste de clones actifs est coûteux en ressources de stockage (ajout de la mémoire *Mca2*). La profondeur de la mémoire contenant cette liste temporaire de clones actifs doit être égale à celle de la mémoire *Mca* contenant les clones actifs utilisés dans le décodage.

G. Extension de l'architecture pour fonctionner avec des mémoires multiples (ARCH06)

Dans toutes les architectures précédentes (ARCH00 à ARCH05), une mémoire unique contenant la matrice est présente. Cette propriété implique que l'accès au contenu de la matrice d'adjacence (initialement stockée dans la mémoire *MM*) est très séquentiel.

Afin de pouvoir accélérer le parcours de la matrice, il est possible réaliser des accès parallèles sur le contenu la matrice en utilisant plusieurs mémoires, chacune contenant une partie de la matrice. L'architecture doit donc être étendue.

La matrice W étant distribuée dans *Nm* mémoires, l'architecture est maintenant composée de *Nm* modules périphérique chacune associée à une partie de cette matrice. Ces modules sont des *Clusters de Calcul* (*CC*) car chacun contient plusieurs *PEs* et ils sont indépendants les uns des autres. Une architecture manipulant ces *Nm* mémoires contient *Nm CCs*. Soit l'ensemble des clusters de calcul d'une architecture, $\mathcal{C}c = \{Cc_0, Cc_1, \dots, Cc_{Nm-1}\}$ .Chaque *CC* contient presque les mêmes éléments que l'architecture de base sauf *Mna* et *Mca* (Voir Figure 0.3).

La mémoire *MM* que possède un *CC* contient une partie de la matrice d'adjacence. Celle-ci contient les 3 zones *ptr*, *asn* et *lignes* et est associée à un sous-ensemble de lignes appartenant à la matrice d'adjacence. Une ligne $Idcl$ (un clone local) sera associée à la mémoire *MM*

contenu dans le cluster de calcul $CC_{Idcl\ MOD\ Nm}$. Cette organisation est similaire à l'attribution des clones (lignes) aux *PEs* (Voir CHAPITRE VI.C.4.c)).

En outre, l'architecture contient un *contrôleur primaire* qui contrôle les clusters de calcul. Il a pour principal rôle de synchroniser les différents *CCs* en particulier en ce qui concerne le score maximal du réseau (récupération du score maximal de chaque *CC*, calcul du score maximal du réseau et diffusion du résultat). Ce maximum doit être obligatoirement calculé grâce à un arbre de comparateurs car tous les *CCs* peuvent envoyer leur score maximal au même moment.

La Figure 0.3 montre l'aperçu d'une telle architecture. Elle comporte de 3 clusters de calcul chacune contenant une mémoire MM et des éléments de calcul.

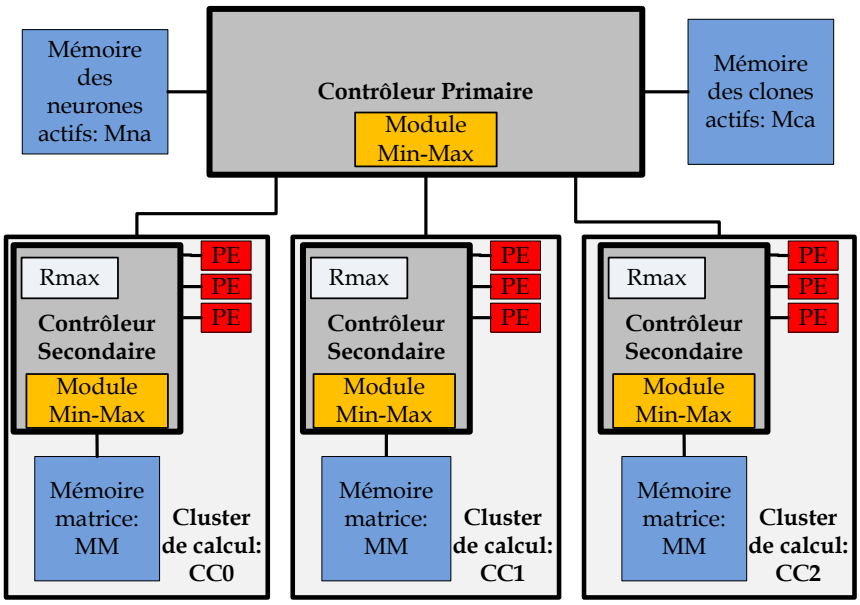


Figure 0.3 Architecture générique multi-mémoire de réseaux à clones

Lors de la phase de pré-calcul PREC, pour initialiser l'état des clones actifs, la liste des clones actifs est à partir de *Mca* et diffusée dans tous les *CCs*. A la réception d'un clone actif *Idcl2*, chaque contrôleur secondaire vérifie s'il est associé à ce clone actif. Si c'est le cas, il va initialiser la zone logique associée à ce clone.

Ensuite, les clones ayant une connexion à 1 avec *Idcl2* sont lus. Pour chaque clone *Idcl1* lu dans un *CC*, sa zone logique est mise à jour de manière identique aux architectures accédant à unique mémoire contenant la matrice (Voir CHAPITRE VI.C).

Si l'architecture contient *Nm* mémoires, alors le nombre maximal de clones connectés lus en parallèle est de *Nm* clones. Le temps requis par l'architecture pour parcourir toute la matrice peut alors au maximum être divisé par *Nm* :

$$T = \frac{Nlig * Nca}{Nm} \text{ cycles} \quad (79)$$

Si l'on considère l'architecture la plus rapide ARCH05 (création dynamique de la liste des clones actifs), alors le temps de calcul global devient :

$$Tc = \frac{Nt}{|PE|} + \frac{T_{CG} * Nlig * Nca}{Nm} + d * Nt + (d * Nt) * \log_2(d * Nt) \text{ cycles} \quad (80)$$

H. Extension de l'architecture générique pour traiter des réseaux ayant une organisation hétéro-associative

L'architecture générique proposée (Voir CHAPITRE VI.C) permet de manipuler des réseaux à clones. Ces réseaux ont une organisation auto-associative. Cependant, il est aussi possible de concevoir des réseaux « hétéro-associatifs » inspirés par les réseaux à clones et de le manipuler avec la même architecture. En effet, des réseaux basés sur le GBNN et intégrant un aspect hétéro-associatif ont déjà été proposés [103].

Un réseau avec une organisation hétéro-associative contient deux couches totalement différentes. Chaque couche contient des clusters et un clone appartenant à un cluster d'une couche ne peut être connecté qu'aux clones faisant partie des clusters de l'autre couche.

Lors de l'apprentissage, le réseau reçoit deux messages, chaque message étant associé à une couche. Pour chaque couche, l'apprentissage consiste alors à : (1) activer les neurones associés aux valeurs de symbole dans le message associé à cette couche, (2) activer les clones associés à ces neurones, (3) sélectionner le sous-réseau et les clones actifs de ce sous-réseau (un par cluster) qui permettront d'apprendre l'association entre les messages et (4) mettre à 1 les poids des connexions entre les clones sélectionnés.

Le décodage dans un réseau hétéro-associatif est différent de celui d'un réseau auto-associatif. Au lieu de se produire simultanément pour tous les clones du réseau, celui-ci se déroule couche après couche. Les valeurs des clones d'une couche ne sont pas calculées au même moment que les valeurs des clones d'une autre couche.

Soit un réseau R contenant deux couches $L^{(0)}$ et $L^{(1)}$.

De manière similaire aux réseaux de Willshaw, Les étapes requises pour calculer les valeurs des clones de la couche $L^{(1)}$ sont les suivantes: (1) calcul du score des clones de la couche $L^{(1)}$ à partir des valeurs des clones de la couche $L^{(0)}$, (2) calcul du score maximal des clones de $L^{(1)}$, (3) mise à jour de la valeur de chaque clone de cette couche.

La condition d'un décodage réussi est similaire à celle des réseaux à clones. Pour qu'un décodage soit réussi, il faut qu'à la fin du décodage, il existe un unique sous-réseau dans chaque couche de telle sorte que chaque cluster de ce sous-réseau ne contienne qu'un unique neurone actif (Voir CHAPITRE III.C.4 pour plus de détails sur le décodage dans les réseaux à clones).

Afin que l'architecture proposée réalise le décodage pour la couche $L^{(1)}$, il faut que la mémoire des clones actifs *Mca* soit initialisée selon un certain ordre. Elle doit contenir la liste des clones actifs de la couche $L^{(1)}$ suivie de la liste des clones actifs de la couche $L^{(0)}$. Cette organisation a deux utilités :

- La liste des clones actifs de $L^{(1)}$ permet d'initialiser les zones logiques des clones actifs (avec allocation de zones logiques pour chaque clone dans le cas d'une allocation dynamique des zones logiques).
- La liste des clones actifs de $L^{(0)}$ permet de parcourir la matrice d'adjacence et de calculer le score des clones de $L^{(1)}$ (sans allocation de zones pour les clones de $L^{(0)}$ dans le cas d'une allocation dynamique des zones logiques).

Les autres phases du décodage restent inchangées (mise à jour des états des clones, mise à jour de la liste des clones actifs de $L^{(1)}$). Une stratégie similaire peut être utilisée pour réaliser le décodage pour la couche $L^{(0)}$ en utilisant les valeurs des clones de la couche $L^{(1)}$.

Bibliographie

[1] E. R. Kandel, H. Markram, P. M. Matthews, R. Yuste, and C. Koch, "Neuroscience thinks big (and collaboratively)," *Nat. Rev. Neurosci.*, vol. 14, no. 9, pp. 659–664, Sep. 2013.

[2] "Grand Challenges - Reverse-Engineer the Brain." [Online]. Available: http://www.engineeringchallenges.org/challenges/9109.aspx. [Accessed: 03-Sep-2015].

[3] B. Widrow, D. E. Rumelhart, and M. A. Lehr, "Neural Networks: Applications in Industry, Business and Science," *Commun ACM*, vol. 37, no. 3, pp. 93–105, Mar. 1994.

[4] J. L. Patel and R. K. Goyal, "Applications of artificial neural networks in medical science," *Curr. Clin. Pharmacol.*, vol. 2, no. 3, pp. 217–226, Sep. 2007.

[5] F. Shadabi, D. Sharma, and R. Cox, "Learning from Ensembles: Using Artificial Neural Network Ensemble for Medical Outcomes Prediction," in *Innovations in Information Technology, 2006*, 2006, pp. 1–5.

[6] F. Amato, A. López, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel, "Artificial neural networks in medical diagnosis," *J. Appl. Biomed.*, vol. 11, no. 2, pp. 47–58, 2013.

[7] C. Liu, J. Liu, F. Yu, Y. Huang, and J. Chen, "Handwritten character recognition with sequential convolutional neural network," in *2013 International Conference on Machine Learning and Cybernetics (ICMLC)*, 2013, vol. 01, pp. 291–296.

[8] B. Baird, M. W. Hirsch, and F. Eeckman, "A neural network associative memory for handwritten character recognition using multiple Chua characters," *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, vol. 40, no. 10, pp. 667–674, Oct. 1993.

[9] J. Du, J.-S. Hu, B. Zhu, S. Wei, and L.-R. Dai, "A Study of Designing Compact Classifiers Using Deep Neural Networks for Online Handwritten Chinese Character Recognition," in *2014 22nd International Conference on Pattern Recognition (ICPR)*, 2014, pp. 2950–2955.

[10] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust Object Recognition with Cortex-Like Mechanisms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 411–426, Mar. 2007.

[11] E. I. Altman, G. Marco, and F. Varetto, "Corporate distress diagnosis: Comparisons using linear discriminant analysis and neural networks (the Italian experience)," *J. Bank. Finance*, vol. 18, no. 3, pp. 505–529, May 1994.

[12] R. C. Lacher, P. K. Coats, S. C. Sharma, and L. F. Fant, "A neural network for classifying the financial health of a firm," *Eur. J. Oper. Res.*, vol. 85, no. 1, pp. 53–65, Aug. 1995.

[13] G. Zhang, M. Y. Hu, B. Eddy Patuwo, and D. C. Indro, "Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis," *Eur. J. Oper. Res.*, vol. 116, no. 1, pp. 16–32, Jul. 1999.

[14] S. T. A. Niaki and S. Hoseinzade, "Forecasting S&P 500 index using artificial neural networks and design of experiments," *J. Ind. Eng. Int.*, vol. 9, no. 1, pp. 1–9, Feb. 2013.

[15] J. M. Nageswaran, M. Richert, N. Dutt, and J. L. Krichmar, "Towards reverse engineering the brain: Modeling abstractions and simulation frameworks," in *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, 2010, pp. 1–6.

[16] C. Mead, "Neuromorphic electronic systems," *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct. 1990.

[17] S. Furber and S. Temple, "Neural systems engineering," *J. R. Soc. Interface*, vol. 4, no. 13, pp. 193–206, Apr. 2007.

[18] J. He, M. G. Maltenfort, Q. Wang, and T. M. Hamm, "Learning from biological systems: modeling neural control," *IEEE Control Syst.*, vol. 21, no. 4, pp. 55–69, Aug. 2001.

[19] T. Giotis, M. A. Christodoulou, and Y. Boutalis, "Identification of combination therapy models using a neuro fuzzy identification scheme," in *2011 19th Mediterranean Conference on Control Automation (MED)*, 2011, pp. 1283–1288.

[20] E. Theodorou and F. J. Valero-Cuevas, "Optimality in neuromuscular systems," *Conf. Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. IEEE Eng. Med. Biol. Soc. Annu. Conf.*, vol. 2010, pp. 4510–4516, 2010.

[21] P. Frost Gorder, "Computer Vision, Inspired by the Human Brain," *Comput. Sci. Eng.*, vol. 10, no. 2, pp. 6–11, Mar. 2008.

[22] H. Jhuang, T. Serre, L. Wolf, and T. Poggio, "A Biologically Inspired System for Action Recognition," in *IEEE 11th International Conference on Computer Vision, 2007. ICCV 2007*, 2007, pp. 1–8.

[23] S.-Y. Lee, "Artificial Brain based on Brain-Inspired Algorithms for Human-Like Intelligent Functions," in *IEEE International Conference on Integration Technology, 2007. ICIT '07*, 2007, pp. nil40–nil41.

[24] "Alzheimer : son origine dans le cerveau découverte," *Medisite*. [Online]. Available: http://www.medisite.fr/a-la-une-alzheimer-son-origine-dans-le-cerveau-decouverte.517632.2035.html. [Accessed: 21-Oct-2015].

[25] J. Hawkins and S. Blakeslee, *On Intelligence*. Times Books, 2004.

[26] J. R. Hughes, "Post-tetanic potentiation," *Physiol. Rev.*, vol. 38, no. 1, pp. 91–113, Jan. 1958.

[27] T. Kohonen, "Self-organization and associative memory," *Self-Organ. Assoc. Mem. 100 Figs XV 312 Pages Springer-Verl. Berl. Heidelb. N. Y. Also Springer Ser. Inf. Sci. Vol. 8*, vol. 1, 1988.

[28] J. Bower and D. Beeman, "Constructing Realistic Neural Simulations with GENESIS," in *Neuroinformatics*, Humana Press, 2007, pp. 103–125.

[29] M. L. Hines and N. T. Carnevale, "NEURON: a tool for neuroscientists," *Neurosci. Rev. J. Bringing Neurobiol. Neurol. Psychiatry*, vol. 7, no. 2, pp. 123–135, Apr. 2001.

[30] D. George and J. Hawkins, "Towards a Mathematical Theory of Cortical Micro-circuits," *PLoS Comput Biol*, vol. 5, no. 10, p. e1000532, Oct. 2009.

[31] M. J. Milford, J. Wiles, and G. F. Wyeth, "Solving Navigational Uncertainty Using Grid Cells on Robots," *PLoS Comput Biol*, vol. 6, no. 11, p. e1000995, Nov. 2010.

[32] J. L. Krichmar, P. Coussy, and N. Dutt, "Large-Scale Spiking Neural Networks Using Neuromorphic Hardware Compatible Models," *J Emerg Technol Comput Syst*, vol. 11, no. 4, pp. 36:1–36:18, Apr. 2015.

[33] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[34] H. Markram, "Seven challenges for neuroscience," *Funct. Neurol.*, vol. 28, no. 3, pp. 145 – 151, 17 2013.

[35] "The Human Brain Project - Human Brain Project." [Online]. Available: https://www.humanbrainproject.eu/. [Accessed: 19-Oct-2015].

[36] "IBM Research: Brain-inspired Chip," 05-Aug-2015. [Online]. Available: http://www.research.ibm.com/articles/brain-chip.shtml. [Accessed: 13-Aug-2015].

[37] "http://apt.cs.man.ac.uk/projects/SpiNNaker."

[38] T. Sharp, R. Petersen, and S. Furber, "Real-time million-synapse simulation of rat barrel cortex," *Neuromorphic Eng.*, vol. 8, p. 131, 2014.

[39] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *IEEE International Joint Conference on Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence)*, 2008, pp. 2849–2856.

[40] "Objectives - Human Brain Project." [Online]. Available: https://www.humanbrainproject.eu/discover/the-project/strategic-objectives. [Accessed: 19-Oct-2015].

[41] A. Calimera, E. Macii, and M. Poncino, "The Human Brain Project and neuromorphic computing," *Funct. Neurol.*, vol. 28, no. 3, pp. 191 – 196, 17 2013.

[42] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, 2011, pp. 1–4.

[43] W. Maas, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Trans Soc Comput Simul Int*, vol. 14, no. 4, pp. 1659–1671, Dec. 1997.

[44] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[45] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proc. Natl. Acad. Sci.*, vol. 81, no. 10, pp. 3088–3092, May 1984.

[46] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw. Publ. IEEE Neural Netw. Counc.*, vol. 14, no. 6, pp. 1569–1572, 2003.

[47] J. Harkin, F. Morgan, S. Hall, P. Dudek, T. Dowrick, and L. McDaid, "Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks," in *International Conference on Field Programmable Logic and Applications, 2008. FPL 2008*, 2008, pp. 483–486.

[48] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[49] M. L. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Mit Press, 1972.

[50] C. V. D. Malsburg, "Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms," in *Brain Theory*, D. G. Palm and D. A. Aertsen, Eds. Springer Berlin Heidelberg, 1986, pp. 245–248.

[51] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[52] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.

[53] R. Hecht-Nielsen, "Neural Networks Letter: Cogent Confabulation," *Neural Netw*, vol. 18, no. 2, pp. 111–115, Mar. 2005.

[54] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-Holographic Associative Memory," *Nature*, vol. 222, no. 5197, pp. 960–962, juin 1969.

[55] B. Graham and D. Willshaw, "Improving recall from an associative memory," *Biol. Cybern.*, vol. 72, no. 4, pp. 337–346, Mar. 1995.

[56] G. Palm, "Neural Associative Memories and Sparse Coding," *Neural Netw*, vol. 37, pp. 165–171, Jan. 2013.

[57] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[58] H. Jarollahi, V. Gripon, N. Onizawa, and W. J. Gross, "Algorithm and Architecture for a Low-Power Content-Addressable Memory Based on Sparse Clustered Networks," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 23, no. 4, pp. 642–653, Apr. 2015.

[59] H. Jarollahi, N. Onizawa, V. Gripon, N. Sakimura, T. Sugibayashi, T. Endoh, H. Ohno, T. Hanyu, and W. J. Gross, "A Nonvolatile Associative Memory-Based Context-Driven Search Engine Using 90 nm CMOS/MTJ-Hybrid Logic-in-Memory Architecture," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 4, no. 4, pp. 460–474, Dec. 2014.

[60] E. Guzmán, O. M. C. Jiménez, A. D. Pérez, and O. Pogrebnyak, "Using Associative Memories for Image Segmentation," in *Advances in Neural Networks – ISNN 2011*, D. Liu, H. Zhang, M. Polycarpou, C. Alippi, and H. He, Eds. Springer Berlin Heidelberg, 2011, pp. 388–396.

[61] R. Danilo, H. Jarollahi, V. Gripon, P. Coussy, L. Conde-Canencia, and W. J. Gross, "Algorithm and implementation of an associative memory for oriented edge detection using improved clustered neural networks," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2501–2504.

[62] V. Gripon and C. Berrou, "Sparse neural networks with large learning diversity," *Neural Netw. IEEE Trans. On*, vol. 22, no. 7, pp. 1087–1096, 2011.

[63] B. Boguslawski, V. Gripon, F. Seguin, and F. Heiztmann, "Huffman Coding for Storing Non-uniformly Distributed Messages in Networks of Neural Cliques," in *AAAI 2014 : the 28th Conference on Artificial Intelligence*, Québec, Canada, 2014, vol. 1, pp. 262–268.

[64] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross, "Architecture and implementation of an associative memory using sparse clustered networks," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 2901–2904.

[65] H. Jarollahi, N. Onizawa, V. Gripon, and W. J. Gross, "Reduced-complexity binary-weight-coded associative memories," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 2523–2527.

[66] B. Kröse and P. van der Smagt, *An Introduction to Neural Networks*. University of Amsterdam, 1996.

[67] G. Palm, "On associative memory," *Biol. Cybern.*, vol. 36, no. 1, pp. 19–31, Feb. 1980.

[68] B. Kosko, "Bidirectional Associative Memories," *IEEE Trans. Syst. MAN Cybern.*, vol. 18, no. 1, pp. 49–60, 1988.

[69] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[70] Q. Qiu, Q. Wu, M. Bishop, R. E. Pino, and R. W. Linderman, "A Parallel Neuromorphic Text Recognition System and Its Implementation on a Heterogeneous High-Performance Computing Cluster," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 886–899, May 2013.

[71] Q. Chen, Q. Qiu, Q. Wu, M. Bishop, and M. Barnell, "A confabulation model for abnormal vehicle events detection in wide-area traffic monitoring," in *2014 IEEE International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, 2014, pp. 216–222.

[72] L. Boltzmann, "Model," in *Theoretical Physics and Philosophical Problems*, B. McGuinness, Ed. Springer Netherlands, 1974, pp. 213–220.

[73] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines*," *Cogn. Sci.*, vol. 9, no. 1, pp. 147–169, 1985.

[74] E. H. L. Aarts and J. H. M. Korst, "Boltzmann machines and their applications," in *PARLE Parallel Architectures and Languages Europe*, J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, Eds. Springer Berlin Heidelberg, 1987, pp. 34–50.

[75] R. G. Morris, "D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949," *Brain Res. Bull.*, vol. 50, no. 5–6, p. 437, Dec. 1999.

[76] B. K. Aliabadi, C. Berrou, V. Gripon, and X. Jiang, "Storing Sparse Messages in Networks of Neural Cliques," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 5, pp. 980–989, mai 2014.

[77] L. Lin, R. Osan, and J. Z. Tsien, "Organizing principles of real-time memory encoding: neural clique assemblies and universal neural codes," *Trends Neurosci.*, vol. 29, no. 1, pp. 48–57, Jan. 2006.

[78] V. Gripon and C. Berrou, "Nearly-optimal associative memories based on distributed constant weight codes," in *Information Theory and Applications Workshop (ITA), 2012*, 2012, pp. 269–273.

[79] D. . Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[80] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.

[81] A. Abubakar Mas'ud, B. G. Stewart, and S. G. McMeekin, "Application of an ensemble neural network for classifying partial discharge patterns," *Electr. Power Syst. Res.*, vol. 110, pp. 154–162, May 2014.

[82] Z.-S. Zhao, X. Feng, Y. Lin, F. Wei, S.-K. Wang, T.-L. Xiao, M.-Y. Cao, and Z.-G. Hou, "Evolved neural network ensemble by multiple heterogeneous swarm intelligence," *Neurocomputing*, vol. 149, Part A, pp. 29–38, Feb. 2015.

[83] B. Girau, "FPNA: Applications and Implementations," in *FPGA Implementations of Neural Networks*, A. R. Omondi and J. C. Rajapakse, Eds. Springer US, 2006, pp. 103–136.

[84] M. Bohrn, L. Fujcik, and R. Vrba, "Field Programmable Neural Array for feed-forward neural networks," in *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, 2013, pp. 727–731.

[85] R. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "An FPGA design framework for large-scale spiking neural networks," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 457–460.

[86] J.-P. Diguet, M. Strum, N. Le Griguer, L. Caetano, and M. J. Sepúlveda, "Scalable NoC-based Architecture of Neural Coding for New Efficient Associative Memories," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Piscataway, NJ, USA, 2013, pp. 19:1–19:9.

[87] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans Math Softw*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011.

[88] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of Sparse Matrix-vector Multiplication on Emerging Multicore Platforms," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2007, pp. 38:1–38:12.

[89] G. Kuzmanov and M. Taouil, "Reconfigurable sparse/dense matrix-vector multiplier," in *International Conference on Field-Programmable Technology, 2009. FPT 2009*, 2009, pp. 483–488.

[90] L. Zhuo and V. K. Prasanna, "Sparse Matrix-Vector Multiplication on FPGAs," in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*, New York, NY, USA, 2005, pp. 63–74.

[91] D. Gregg, C. Mc Sweeney, C. McElroy, F. Connor, S. McGettrick, D. Moloney, and D. Geraghty, "FPGA Based Sparse Matrix Vector Multiplication using Commodity DRAM Memory," in *International Conference on Field Programmable Logic and Applications, 2007. FPL 2007*, 2007, pp. 786–791.

[92] R. Dorrance, F. Ren, and D. Marković, "A Scalable Sparse Matrix-vector Multiplication Kernel for Energy-efficient Sparse-blas on FPGAs," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, New York, NY, USA, 2014, pp. 161–170.

[93] "cuBLAS," 25-Sep-2014. [Online]. Available: https://developer.nvidia.com/cublas. [Accessed: 15-Oct-2015].

[94] "cuSPARSE," 25-Sep-2014. [Online]. Available: https://developer.nvidia.com/cusparse. [Accessed: 15-Oct-2015].

[95] N. Bell and M. Garland, "Implementing Sparse Matrix-vector Multiplication on Throughput-oriented Processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009, pp. 18:1–18:11.

[96] T. Nechma and M. Zwolinski, "Parallel Sparse Matrix Solution for Circuit Simulation on FPGAs," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1090–1103, Apr. 2015.

[97] Y. Wang, H. Yan, C. Pan, and S. Xiang, "Image editing based on Sparse Matrix-Vector multiplication," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 1317–1320.

[98] S. Kestur, J. D. Davis, and E. S. Chung, "Towards a Universal FPGA Matrix-Vector Multiplication Architecture," in *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, Washington, DC, USA, 2012, pp. 9–16.

[99] "QDR-II | Cypress." [Online]. Available: http://www.cypress.com/products/qdr-ii. [Accessed: 24-Oct-2015].

[100] C. Chavet, P. Coussy, and N. Charpentier, "Neural network architecture, production method, and programmes corresponding thereto," WO/2014/079990, 30-May-2014.

[101] V. Gripon and C. Berrou, "Nearly-optimal associative memories based on distributed constant weight codes," in *Information Theory and Applications Workshop (ITA), 2012*, 2012, pp. 269–273.

[102] P. Coussy, C. Chavet, H. Wouafo, and L. Conde-Canencia, "Fully Binary Neural Network Model and Optimized Hardware Architectures for Associative Memories," *J Emerg Technol Comput Syst*, vol. 11, no. 4, pp. 35:1–35:23, avril 2015.

[103] R. Danilo, P. Coussy, L. Conde-Canencia, V. Gripon, and W. J. Gross, "Restricted Clustered Neural Network for Storing Real Data," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, New York, NY, USA, 2015, pp. 205–210.

[104] Z. Yao, V. Gripon, and M. Rabbat, "A GPU-based associative memory using sparse Neural Networks," in *2014 International Conference on High Performance Computing Simulation (HPCS)*, 2014, pp. 688–692.

[105] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 26-Oct-2015].

[106] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction à l'algorithmique*. Dunod, 1994.

Liste des publications

Articles en revue internationale

P. Coussy, C. Chavet, **H. Wouafo** and L. C. Canencia, "Fully-Binary Neural Network Model and Optimized Hardware Architectures for Associative Memories," *ACM J. Emerg. Technol. Comput. Syst.*, Sep. 2014.

Conférences internationales

H. Wouafo, C. Chavet, and P. Coussy, "Improving Storage of Patterns in Recurrent Neural Networks: Clone Based Model and Architecture," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, Lisbon (Portugal), 2015.

Conférences nationales

H. Wouafo, C. Chavet and P. Coussy, "Modèle et Architecture de Réseaux de Neurones Récurrents à Clones", In Colloque National du GRETSI, Lyon, France, september 2015.

Résumé

De nos jours, les réseaux de neurones artificiels sont largement utilisés dans de nombreuses applications telles que le traitement d'image ou du signal. Récemment, un nouveau modèle de réseau de neurones a été proposé pour concevoir des mémoires associatives, le GBNN (Gripon-Berrou Neural Network). Ce modèle offre une capacité de stockage supérieure à celle des réseaux de Hopfield lorsque les informations à mémoriser ont une distribution uniforme. Des méthodes améliorant leur performance pour des distributions non-uniformes ainsi que des architectures matérielles mettant en œuvre les réseaux GBNN ont été proposées. Cependant, ces solutions restent très coûteuses en ressources matérielles, et les architectures proposées sont restreintes à des réseaux de tailles fixes et sont incapables de passer à l'échelle.

Les objectifs de cette thèse sont les suivants : (1) concevoir des modèles inspirés du modèle GBNN et plus performants que l'état de l'art, (2) proposer des architectures moins coûteuses que les solutions existantes et (3) concevoir une architecture générique configurable mettant en œuvre les modèles proposés et capable de manipuler des réseaux de tailles variables.

Les résultats des travaux de thèse sont exposés en plusieurs parties. Le concept de *réseaux à clones de neurone* et ses différentes instanciations sont présentés dans un premier temps. Ces réseaux offrent de meilleures performances que l'état de l'art pour un coût mémoire identique lorsqu'une distribution non-uniforme des informations à mémoriser est considérée. Des optimisations de l'architecture matérielle sont ensuite introduites afin de fortement réduire le coût en termes de ressources. Enfin, une architecture générique capable de passer à l'échelle et capable de manipuler des réseaux de tailles variables est proposée.

Numéro d'ordre : 394